

공학박사학위논문

딥 러닝을 활용한 한국 아파트 단위평면  
분석 방법론 개발

Deep Learning Based Spatial Analysis Method for Korean  
Apartment Unit Plans

2021년 2월

서울대학교 대학원

건축학과

안의순



# 딥 러닝을 활용한 한국 아파트 단위평면 분석 방법론 개발

지도교수 최 재 필

이 논문을 공학박사 학위논문으로 제출함

2021년 1월

서울대학교 대학원

건축학과

안의순

안의순의 공학박사 학위논문을 인준함

2020년 12월

위 원 장	최 병 희	(인)
부위원장	최 재 편	(인)
위 원	조 향 만	(인)
위 원	김 성 애	(인)
위 원	박 정 대	(인)



## 초록

이 연구에서는 한국 아파트 단위평면에서 나타나는 주거공간의 배치를 분석할 수 있는 방법론을 딥 러닝 방법론을 활용하여 개발하였다. 단위평면에 대한 정량화된 분석을 자동으로 수행하여 한국 아파트 전수에 대한 분석이 가능한 방법론을 개발하고자 하였다.

한국에서 아파트가 주거의 주류가 된 이후, 폭발적으로 늘어난 아파트 전수에 대한 연구가 어려워지면서, 단위평면에 대한 연구는 다양한 기준에 따라 파편화되었다. 공간구문론을 포함한 공간분석은 건축 공간의 구조에 대한 정량적인 분석이 가능한 방법론으로, 한국 아파트의 주거 공간을 객관적이고 재현 가능하게 분석할 수 있는 기반을 제공하였다. 그러나 공간분석 방법론에서도 개별 사례마다 수작업이 필요하다는 한계로 인하여, 1990년대 말 이후로는 한국 아파트 전수에 대한 공간분석 연구는 이루어지지 않았다. 이러한 한계를 극복하기 위하여, 연구자의 의도를 스스로 데이터에서 귀납적으로 학습하고 이를 한국 아파트 전수에 적용하여 분석을 수행할 수 있는 방법론을 개발하고자 하였다. 이러한 분석 방법론을 개발하기 위하여, 이 연구에서는 기계학습 방법론 중 하나인 딥 러닝을 활용하였다. 딥 러닝은 심층 신경망 모형을 활용하여 복잡한 개념을 처리할 수 있다는 장점이 있다.

2장에서는 건축 공간에 딥 러닝 방법론을 적용한 선행 연구를 고찰하여, 건축 공간을 2차원 행렬 형태로 표현하여 CNN (convolutional neural network, 합성곱 신경망) 모형을 성공적으로 적용할 수 있음을 확인하였다. 또한, 딥 러닝 모형이 건축 공간 데이터에서 학습한 규칙은 잠재공간 분석이나 역합성곱 신경망 학습 등의 방법론을 통하여 분석이 가능함을 파악하였다. 그러나, 다른 건축 공간을 대상으로 학습한 딥 러닝 모형을 한국 아파트에 그대로 적용할 수는 없었다. 따라서 한국 아파트에 딥 러닝을 적용하여 분석하기 위해서는 한국 아파트 전수에 대한 데이터셋을 구축하는 것이 필요함을 확인하였다.

3장에서는 딥 러닝을 활용하여 한국 아파트 단위평면의 주거공간 배치를 분석하는 방법론을 수립하였다. 먼저, 공간분석과 딥 러닝에 대한 고찰을 반영하여, 건축 공간을 2차원 격자 구조로, 각 지점의 주거공간 특성을 깊이 차원으로 재현한 2차원 이미지 형태로 주거공간 배치를 모형화하였다. 다음으로, 딥 러닝 모형이 학습한 주거공간 배치유형과 단위평면의 각 영역 사이의 관계를 시각화할 수 있는 CAM (class activation mapping) 방법론을 주거공간 배치 특성 분석을 위한 방법론으로 선정하였다. 이를 통하여, 각 유형별 단위평면의 주거공간 배치에서 나타나는 계획 특성을 시각화할 수 있다.

CAM 방법론은 연구자가 사전에 설정한 분류 기준에 따른 비교만 가능한 한계가 있다. 이 연구에서는 CAM 방법론을 확장하여, 딥 러닝 모형이 학습한 주거공간 배치 특성을 기준으로 평면 사례를 귀납적으로 유형화하고 그 특성을 시각화하는 BAM (bicluster activation map) 방법론을 수립하였다. BAM 방법론은, 바이클러스터링 분석을 통하여 딥 러닝 모형 내부의 잠재표현 계층의 각 노드와 이를 활성화하는 단위평면 사례를 짹지어 유형화한 주거공간 배치유형을 도출하고, 단위평면의 각 영역에 대하여 해당 배치유형의 모형이 활성화되는 영역을 시각화한다.

또한, 이렇게 개발된 단위평면 분석 방법론을 한국 아파트 단위평면의 진화과정 분석에 적용하여 검증하기 위한 방법론을 수립하였다. 한국 아파트 단위평면 데이터셋에 대한 시기 구분 작업을 학습한 딥 러닝 모형을 활용하여, 시기 구분과 관련된 주거공간 배치 특성을 기준으로 단위평면을 유형화하고, 그 분석 결과를 한국 아파트 선행 연구와 비교하여 분석 방법론의 실효성을 확인하고자 하였다.

4장에서는 한국 아파트 단위평면의 진화과정 분석과 이를 통한 분석 방법론 검증에 필요한 한국 아파트 단위평면 데이터셋을 구축하였다. 공개된 단위평면 및 연관 자료를 수집하고, 수집된 평면도에서 현관, LDK (거실 및 주방, 식당), 침실, 발코니, 화장실 등 주거공간의 배치를 인식하여, 한국 아파트 전수에

대한 데이터셋을 구축할 수 있었다.

일관된 분석을 위하여 주 향 및 현관의 방향, 축척이 통일되도록 정규화하고, 서로 다른 크기의 평면에 대하여 동일한 분석 영역을 설정하였다. 준공 시기, 시도별 권역 등 자료도 딥 러닝 모형의 입력값으로 적합하도록 정규화하였다. 특히 준공 시기는 여러 선행 연구에서 5년 또는 10년 단위로 시기를 구분하는 것을 반영하여, 수집된 단위평면 사례의 준공연도 범위인 1969년부터 2019년 까지 51년을 5년 단위 10개 시기로 설정하였다. 이러한 과정을 거쳐, 약 5만 개 단위평면과 연관 자료를 데이터셋으로 구축하였다.

5장에서는 이 연구에서 개발한 분석 방법론을 적용하여 한국 아파트의 단위평면에서 나타나는 시기에 따른 주거공간 배치 변화를 분석하였다. 먼저, 3장에서 선정한 딥 러닝 모형인 VGG-GAP 모형을 4장에서 구축한 한국 아파트 단위평면 데이터셋에 맞게 수정한 후, 단위평면을 시기에 따라 분류하는 작업을 수정된 모형에 학습시켰다. 이러한 과정을 거쳐, 주거공간 배치 분석 모형은 단위평면 50,252개 중 29,027개 평면에 대해서는 5년 단위 시기로 정확히 예측하고, 16,457개 평면은 직전 시기 또는 직후 시기로 예측하여, 총 90.51%의 단위평면에 대하여 5년 이내 오차로 준공 시기를 예측할 수 있음을 확인하였다.

다음으로, 딥 러닝 모형 내부의 잠재표현 계층에 학습된 시기별 주거공간 배치 특성을 3장에서 수립한 BAM 방법론을 통하여 분석하였다. 약 5만 개 평면과 512개 잠재표현 노드에 대한 바이클러스터링 분석을 통하여 16개 주거공간 배치유형을 도출하였다. 각 배치유형은 준공 시기를 기준으로 평면 사례와 이에 활성화되는 잠재표현 노드를 함께 군집화하였다. 그러나 인접한 시기의 유형 사이에서 활성화 강도의 상관관계가 높게 나타나, 배타적인 1:1 관계로 해석하기 어려운 특성도 확인하였다.

각 배치유형에 속한 단위평면 사례는 시기별로 다양하게 분포되어 나타났다. 그러나 지역별, 규모별로는 일부 유형을 제외하면 큰 차이를 보이지 않았다.

BAM 시각화를 통하여 각 배치유형별 단위평면 사례에서 해당 유형별 모형이 활성화되는 영역을 분석한 결과, 주거공간 배치유형이 여러 시기의 한국 아파트 단위평면에서 나타나는 다양한 계획 특성을 학습하였음을 확인할 수 있었다.

6장에서는 5장에서 도출된 주거공간 배치유형을 주동 형식별 단위평면 사례에 적용하여 한국 아파트의 진화과정을 해석하고, 그 결과를 선행 연구와 비교하였다. 이러한 과정을 통하여, 이 연구에서 개발한 분석 방법론을 검증하고, 궁극적으로는 딥 러닝을 활용한 귀납적 분석 방법론을 건축 공간 연구에 활용할 수 있는 가능성을 검토하였다.

1980년대부터 2010년대까지, 판상형과 혼합형 주동의 복도형, 계단실형 평면, 탑상형 주동의 중앙부 및 단부 평면, 원룸형 도시형생활주택 평면 등 다양한 한국 아파트 단위평면 사례를 대상으로, 해당 시기의 여러 배치유형별 모형의 활성화 영역을 BAM 시각화를 적용하여 분석하였다.

이러한 분석을 통하여, 주거공간 배치유형이 1990년대 판상형 주동의 계단실형 단위평면에서 나타나는 주거공간 배치 특성을 잘 학습하였음을 확인하였다. 또한, BAM 시각화를 통하여 복도형 평면이나 탑상형 주동의 평면 등에서 나타나는 고유한 주거공간 배치를 도출할 수 있었다. 그러나 주거공간 배치에 명확하게 구분할 수 있는 차이점이 있는 경우 그 차이점에 집중하여 전반적인 주거공간 배치를 무시하는 한계를 보였다.

이러한 분석을 통하여 도출된 한국 아파트의 진화과정은 계단실형 주동의 일반화, 판상형 주동에서 소수 유형으로의 수렴, 대형에서 소형 평면으로의 하향화(trickle-down), 탑상형 주동의 도입, 발코니 면적 극대화 및 발코니 확장의 일반화 등 선행 연구의 발견과 대체로 일치하였다. 이러한 결과를 통하여, 이 연구에서 개발한 단위평면 분석 방법론을 통한 한국 아파트의 진화과정 분석이 가능함을 확인하였다.

이 연구에서 개발한 단위평면 분석 방법론은 개별 사례에 대한 연구자의 수작

업이 필요하지 않기 때문에, 한국 아파트에 대한 전수조사가 가능하다는 점에 가장 큰 의의가 있다. 또한, 주거공간 배치와 준공 시기만 주고 학습한 딥 러닝 모형이 한국 아파트의 다양한 평면 유형을 인식할 수 있음을 보임으로써, 이 연구의 방법론이 평면 계획과 직접적인 관련이 없는 추상적인 기준에 따른 분석도 잘 수행할 수 있음을 확인하였다. 이 연구를 통하여, 한국 아파트의 주거공간 배치를 다양한 기준으로 분석하고, 아파트가 아닌 다른 건축 공간에 대하여 분석 대상을 확장하며, 단위평면의 생성적 설계 방법론을 개발하는 등, 후속 연구를 위한 이론적 토대를 구축하였다.

**주요어:** 공동주택, 단위세대, 공간구조, 기계학습, 딥 러닝, 심층 신경망 학습, 유형학

**학번:** 2011-30177



# 목차

초록	i
<b>1 서론</b>	<b>1</b>
1.1 연구의 배경 . . . . .	1
1.2 연구의 목적 . . . . .	5
1.3 연구의 방법 . . . . .	6
<b>2 건축 공간과 딥 러닝</b>	<b>9</b>
2.1 기계학습 . . . . .	9
2.2 딥 러닝 . . . . .	14
2.3 건축 공간에 대한 딥 러닝 . . . . .	18
2.4 딥 러닝의 해석 . . . . .	23
2.5 소결 . . . . .	27
<b>3 딥 러닝을 활용한 단위평면 분석 방법론</b>	<b>31</b>
3.1 건축 공간 모형화 . . . . .	32
3.2 표현학습을 통한 주거공간 배치 분석 . . . . .	37
3.3 딥 러닝을 적용한 단위평면 유형화 . . . . .	41
3.4 방법론 검증 . . . . .	48

<b>4 한국 아파트 단위평면 데이터셋</b>	<b>51</b>
4.1 단위평면 자료 수집 . . . . .	53
4.2 주거공간 배치 인식 . . . . .	54
4.2.1 단위평면 분석 전략 . . . . .	56
4.2.2 단위세대 및 실 영역 인식 . . . . .	59
4.3 평면도 정규화 . . . . .	63
4.3.1 평면도 방향 정규화 . . . . .	66
4.3.2 평면도 축척 정규화 . . . . .	72
4.4 분석 영역 설정 . . . . .	74
4.5 평면유형 연관 데이터 처리 . . . . .	76
4.5.1 준공연도 정규화 . . . . .	76
4.5.2 지역 정규화 . . . . .	78
4.5.3 전용면적 정규화 . . . . .	80
4.5.4 침실 및 화장실 수 . . . . .	82
4.6 단위평면 데이터셋 구축 . . . . .	83
<b>5 한국 아파트 단위평면 분석</b>	<b>85</b>
5.1 시기별 주거공간 배치 분석 모형 . . . . .	85
5.2 주거공간 배치유형 도출 . . . . .	89
5.3 주거공간 배치유형별 계획 특성 분석 . . . . .	97
<b>6 주거공간 배치유형을 통한 한국 아파트 단위평면의 진화과정 해석</b>	<b>125</b>
<b>7 결론</b>	<b>143</b>
<b>참고문헌</b>	<b>149</b>
<b>부록</b>	<b>157</b>
Source code . . . . .	157

10_naver_floorplan_analysis.py . . . . .	157
11_floorplan_normalization.py . . . . .	180
12_floorplan_image.py . . . . .	206
15_process_floorplan_images.py . . . . .	211
16_processed_list.py . . . . .	217
17_image_size.py . . . . .	220
24_dataset_all_train_test_56.py . . . . .	223
41_visualize_floorplan.py . . . . .	231
61_VGG_CAM.py . . . . .	236
62_VGG_CAM_prediction.py . . . . .	243
63_VGG_CAM_activation.py . . . . .	247
65_biclustering.py . . . . .	252
67_VGG_CAM_visualization.py . . . . .	265
<b>Abstract</b>	<b>285</b>



# 그림 목차

2.1	인공지능, 기계학습, 딥 러닝의 관계 . . . . .	10
2.2	다양한 기계학습 모형 비교 . . . . .	11
2.3	가시성 그래프와 딥 러닝 모형 비교 . . . . .	13
2.4	심층 신경망 모형의 구조 . . . . .	15
2.5	알파고에 사용된 합성곱 신경망 . . . . .	16
2.6	딥 러닝의 성장 속도 . . . . .	19
2.7	Pix2Pix . . . . .	20
2.8	ArchiGAN . . . . .	21
2.9	ArchiGAN을 한국 아파트 단위세대에 적용한 결과 . . . . .	22
2.10	교통 표지판 인식 딥 러닝 모형에 대한 적대적 공격 . . . . .	25
2.11	잠재공간 안에 재현된 미술 양식 . . . . .	26
2.12	역합성곱 신경망으로 시각화한 딥 러닝의 이미지 인식 . . . . .	28
3.1	CAM을 통한 분류 근거 영역 시각화 . . . . .	40
3.2	CAM 시각화의 구조 . . . . .	42
3.3	단위평면 CAM 시각화 구조 . . . . .	44
3.4	바이클러스터링 분석 예시 . . . . .	46
4.1	평면도 예시 . . . . .	54
4.2	시도별 연도별 평면도 수록 건수 . . . . .	55

4.3	단위평면 오류 예시 . . . . .	57
4.4	단위평면 실외 영역 인식 . . . . .	60
4.5	단위평면 실 경계 후보 인식 . . . . .	62
4.6	단위평면 실내 영역 인식 . . . . .	64
4.7	단위평면 주거공간 배치 인식 . . . . .	65
4.8	전면과 측면이 개방된 탑상형 주동 평면 사례 . . . . .	68
4.9	단위평면의 폭 및 깊이 분포 . . . . .	74
4.10	전용면적 분포 . . . . .	81
5.1	수정 VGG-GAP 모형 개념도 . . . . .	86
5.2	수정 VGG-GAP 모형 개요 . . . . .	87
5.3	준공 시기 예측 결과 . . . . .	88
5.4	잠재표현 활성화 행렬 . . . . .	90
5.5	바이클러스터 수 결정 . . . . .	91
5.6	잠재표현 활성화 행렬 바이클러스터링 . . . . .	93
5.7	유형별 모형의 평균 활성화 강도 . . . . .	95
5.8	유형별 모형의 활성화 강도에 대한 상관분석 . . . . .	96
5.9	배치유형별 단위평면 시기 분포 . . . . .	98
5.10	배치유형별 단위평면 지역 분포 . . . . .	99
5.11	배치유형별 단위평면 규모 분포 . . . . .	100
5.12	배치유형별 단위평면 침실 수 분포 . . . . .	101
5.13	배치유형별 단위평면 화장실 수 분포 . . . . .	102
5.14	배치유형별, 규모별 단위평면 BAM 시각화 . . . . .	104
5.15	8090-1 유형의 주거공간 배치 계획 특성 . . . . .	105
5.16	8090-2 유형의 주거공간 배치 계획 특성 . . . . .	107
5.17	8090-3 유형의 주거공간 배치 계획 특성 . . . . .	108
5.18	9000-1 유형의 주거공간 배치 계획 특성 . . . . .	109

5.19 9000-2 유형의 주거공간 배치 계획 특성 . . . . .	111
5.20 9000-3 유형의 주거공간 배치 계획 특성 . . . . .	112
5.21 9000-4 유형의 주거공간 배치 계획 특성 . . . . .	113
5.22 00-1 유형의 주거공간 배치 계획 특성 . . . . .	114
5.23 00-2 유형의 주거공간 배치 계획 특성 . . . . .	115
5.24 00-3 유형의 주거공간 배치 계획 특성 . . . . .	116
5.25 00-4 유형의 주거공간 배치 계획 특성 . . . . .	117
5.26 0010-1 유형의 주거공간 배치 계획 특성 . . . . .	119
5.27 0010-2 유형의 주거공간 배치 계획 특성 . . . . .	120
5.28 0010-3 유형의 주거공간 배치 계획 특성 . . . . .	121
5.29 10-1 유형의 주거공간 배치 계획 특성 . . . . .	122
5.30 10-2 유형의 주거공간 배치 계획 특성 . . . . .	123
 6.1 복도형 중형 평면에 대한 유형별 모형의 활성화 영역 . . . . .	129
6.2 복도형 대형 평면에 대한 유형별 모형의 활성화 영역 . . . . .	130
6.3 계단실형 중형 평면에 대한 유형별 모형의 활성화 영역 . . . . .	132
6.4 계단실형 대형 평면에 대한 유형별 모형의 활성화 영역 . . . . .	133
6.5 2000년대 계단실형 평면에 대한 유형별 모형의 활성화 영역 . . . . .	135
6.6 탑상형 중앙부 평면에 대한 유형별 모형의 활성화 영역 . . . . .	138
6.7 탑상형 단부 평면에 대한 유형별 모형의 활성화 영역 . . . . .	140
6.8 혼합형 주동 평면에 대한 유형별 모형의 활성화 영역 . . . . .	141
6.9 원룸형 도시형생활주택 평면에 대한 유형별 모형의 활성화 영역	142



# 표 목차

4.1 시기 구분 정의 . . . . .	77
4.2 시도별 권역 정의 . . . . .	80
4.3 한국 아파트 단위평면 데이터셋 구성 . . . . .	83
5.1 배치유형별 평균 준공연도 . . . . .	91
5.2 배치유형별 계획 특성 . . . . .	124
6.1 분석 대상 한국 아파트 단위평면 사례 . . . . .	128



# 제 1 장

## 서론

### 1.1 연구의 배경

아파트는 한국 도시의 지형을 결정하는 주거 유형이다. 인구 증가와 수도권 집중으로 인한 주택 부족을 해결하기 위하여 1천만 호가 넘게 공급된 아파트는 전체 주택 중 60%를 차지한다.<sup>1</sup> 아파트에 처음으로 살게 된 세대에게 아파트는 ‘닭장’이었고 ‘성냥갑’이었다. ‘아파트 공화국’은 아파트 중심의 한국 도시에 대한 비판적 인식을 반영하는 자화상이자 탈피해야 할 오명으로 언급되었다.<sup>2</sup>

그러나 그로부터 20년이 지나 주택보급률 100%를 달성한 현재도 아파트에 대한 선호는 여전히 높으며, 젊은 층으로 갈수록 더욱 강해 30대 가구의 73%가 아파트 거주를 희망한다.<sup>3</sup> 이러한 선호는 지금까지 지어진 아파트가 앞으로도 리모델링과 재건축을 통해 한국인의 주거를 정의할 ‘정해진 미래’라는 점을 보여준다. 따라서 한국 아파트에 대한 이해는 한국의 미래 주거를 만들어나가기 위한 선결조건이다.

<sup>1</sup>통계청. (2018). 주택의 종류별 주택-읍면동(2015), 시군구(2016~).

<sup>2</sup>이제훈, 이종규. (1999). 마침내 ‘아파트 공화국’. 한겨레신문, 17.

<sup>3</sup>국토교통부. (2016). 2016년도 주거실태조사 통계보고서., pp. 251-252

이질적인 주거 유형이었던 아파트는 한국에서 독자적인 주거 공간을 만들어내며 성공적으로 정착하였다. 이러한 성공은 다른 나라에서는 찾아보기 어려운 한국 아파트만의 특수성이다. 근대화와 도시로의 이주 등 한국 사회가 겪은 변화는 아파트라는 주거 유형의 도입에 큰 영향을 주었다. 그러나 근대화 과정에서 한국만 그러한 변화를 겪은 것은 아니기에, 이러한 원인만으로는 한국에서 아파트가 가장 선호하는 주거가 된 것을 설명하기 어렵다.

공간분석 (spatial analysis) 연구는 한국 아파트의 공간구조에서 그 성공을 설명할 수 있는 요인을 발견하였다. 전상인은 여러 선행연구를 종합하여 한국 아파트가 “현대 한국인의 마음을 사로잡는” 요인 중 하나로 아파트 단위세대 공간구조의 사회문화적 토착화를 들면서, 전통 주거의 중정, 마당, 헛간 등의 공간을 거실, ‘베란다’, 다용도실 등을 통해 잘 담아내었을 뿐 아니라, 한국 사회의 구조적 변동에도 빠르게 대응하여 진화하였다고 설명하였다.<sup>4</sup>

한국 아파트의 거실 중심형 배치는 한국에서 아파트가 도입되면서 가장 먼저 일어난 변화 중 하나이다. 최재필은 공간구문론 (space syntax)과 시각적 접근/노출 (visual access/exposure) 방법론을 통해 도시한옥과 한국 아파트의 공간구조를 비교하였다. 공간구조와 시각적 위계 두 가지 측면에 대한 비교를 통하여, 도시한옥에서 주거 공간의 중심에 위치한 중정의 위상과 역할을 한국 아파트의 거실이 이어받았다는 것을 밝혔다.<sup>5</sup> 서경욱은 방 단위 공간 연결 그래프를 분석하여 1970년대부터 1990년대까지의 한국 아파트에서 공통되는 실내 공간 배치의 유형을 추출하였다. 이를 통해 도시한옥의 중정은 단순히 거실로 바뀐 것이 아니라, 중심 공간으로서의 기능만 하나로 연결된 LDK의 “중앙 홀”이 가져가고 “바닥이 낮고 더러운 공간”으로서의 중정의 기능은 물리적 조건이 유사한 발코니로 옮겨갔음을 발견하였다.<sup>6</sup> 이러한 과정을 거쳐 한국

<sup>4</sup>전상인. (2007). 아파트 선호의 문화사회학. *환경논총*, 45, 11-32.

<sup>5</sup>최재필. (1990). 중정에서 거실로-아파트 대량생산에 있어서의 거주성 확보를 위한 시험적 고찰. *주택도시*, 51, 52-64.

<sup>6</sup>Seo. (2007). Space puzzle in a concrete box: finding design competence that

아파트의 단위평면은 1990년대에 이미 “소수의 사회적으로 합의된 해답으로 수렴”<sup>7</sup>하였다는 것을 정량적으로 확인할 수 있었다.

그 이후로도 한국 아파트의 진화는 멈추지 않았다. 시간이 지나면서 입식 주방이 등장하는 등 생활양식이 변화하고, 한국 주거에서도 “바닥이 낮고 더러운” 공간의 필요성이 줄어들었다. 이러한 변화는 도시한옥의 중정, 대청, 부엌을 통합한 공간 전체에 전통 한옥의 온돌과 마루가 결합한 온돌마루 바닥이 적용되는 형태로 나타났다.<sup>8</sup> 한국 아파트가 전통 주거의 공간을 계승하여 재창조하는 과정은 현재도 계속 이어지고 있는 것이다.

그러나 한국에서 아파트가 주거의 주류가 된 1990년대 말 이후, 한국 아파트 단위평면에 대한 연구는 오히려 대상 지역, 건설 시기, 평면 규모, 설계 주체 등에 따라 파편화되었다. 아파트가 한국인이 가장 선호하는 주거가 되면서, 신축 주거의 대부분이 아파트로 건설되어 폭발적으로 늘어났기 때문에, 소수의 연구자가 수작업을 통하여 전국 아파트에 대한 분석을 수행하기 어려워진 것이다. 정량적인 방법론은 연구자에 따라 달라지지 않는 객관적인 결과를 도출할 수 있다. 그러나 그러한 방법론도 개별 사례마다 수작업을 통한 분석을 요구하는 한, 연구자들의 역량은 파편화되고 전국의 아파트를 대상으로 다양한 시각에서 수행된 분석을 비교할 수 있는 기회를 얻기 어렵게 된다.

이러한 상황은 아파트 공간에 대한 전수조사를 가능하게 하는 새로운 방법론의 필요성을 드러낸다. 그러기 위하여 필요한 방법론은 무엇보다도 연구자의 수작업 없이 분석이 가능하여야 한다. 정량화된 분석을 자동으로 수행하고, 그 결과를 다른 연구자가 재현할 수 있는 방법론은 한국 아파트 전반에 걸친 연구를 가능하게 하고, 한국 주거의 주류가 된 한국 아파트에 대한 여러 연구자의

---

generates the modern apartment houses in Seoul. Environment and Planning B: Planning and Design, 34(6), 1071–1084.

<sup>7</sup>Seo. (2007). Space puzzle in a concrete box: finding design competence that generates the modern apartment houses in Seoul.

<sup>8</sup>전봉희, 권용찬. (2012). 한옥과 한국 주택의 역사. 동녘., pp. 197–200

시각을 비교하도록 하는 도구가 될 수 있다.

공간분석은 주거공간과 그 안에서 일어나는 행태를 모형화하고 정량적으로 분석할 수 있는 기반을 제공하였다. 공간분석의 배경에는 많은 계산을 빠르게 수행할 수 있는 컴퓨터의 등장이 있었다. 컴퓨터의 계산능력이 발전함에 따라, 연구자들은 공간분석 모형 안에 공간과 사회의 상호작용이라는 복잡한 현상을 더욱 잘 담아낼 수 있게 되었다. 이러한 변화는 신도시 건설 등 아파트 보급이 가속화되던 시기와 겹쳐 일어났으며, 연구자들은 진화하는 도구를 활용하여 동시대에서 아파트의 발전 과정을 추적하였다.

최근에는 GPU 등을 활용한 이기종 컴퓨팅 (heterogeneous computing)이 대중화되면서 다양한 분야에서 기계학습 (machine learning)을 적용한 연구가 수행되고 있다. 기계학습은 귀납적으로 데이터에 잠재된 규칙을 찾아낸다는 점이 전통적인 분석 방법론과 다르다. 공간분석에서 공간 구조가 사회 현상에 미치는 영향을 분석하기 위하여 단위공간, 그래프 구조, 중심성 지표 등을 사람이 선형적인 직관을 통해 개발한다면, 기계학습에서는 컴퓨터가 공간 구조와 사회 현상의 수많은 사례를 학습하여 예측 모형을 내놓는 방식이다.

기계학습 방법론 중에서도 특히 딥 러닝 (deep learning, 심층 신경망 학습)은 자연어나 음성, 영상 등 과거에는 정성적 자료로 인식되었던 비정형 데이터 분석에 두각을 나타낸다. 이러한 데이터를 수작업을 거치지 않고 처리할 수 있게 됨으로써 사람이 처리할 수 없는 규모의 데이터에 대한 분석이 가능하였다. 또한, 공간분석과 마찬가지로 분석 과정에서 주관적인 판단이 개입되지 않기 때문에 그 결과가 재현가능하다.

“이론의 종말 (the end of theory)”이라는 표현은 빅데이터와 기계학습이 연구 방법론에 미칠 영향을 보여준다.<sup>9</sup> 현상 자체를 측정하고 분석하여 예측할

---

<sup>9</sup>Anderson. (2008). The end of theory: The data deluge makes the scientific method obsolete. Wired magazine.

수 있게 된다면, 그 현상을 사람이 이해하고 예측하기 위하여 필요했던 분석의 틀로서의 이론의 필요성이 사라진다는 주장이다. 예를 들어 공간분석에서는 공간 안에서 사람들이 보이는 행태를 예측하기 위하여 공간 구조를 재현하는 볼록공간도와 축선도, 가시성 그래프 등의 모형을 개발하였다. 이러한 모형은 공간이 지니는 복잡한 구조의 한 측면만을 보여줄 수 있다. 그런데 만약 공간 그 자체를 인간 행태와 직접 비교하고 분석하는 것이 가능하다면, 공간 구조의 한 측면만을 보여주는 모형에 의존할 필요가 없어진다. 기계학습은 데이터를 가장 잘 설명할 수 있는 모형을 자동으로 도출할 수 있기 때문이다.

이러한 기계학습의 장점은 한국 아파트에 대한 분석에서 더욱 빛날 수 있다. 단지마다, 평면유형마다 조금씩 다른 단위세대 평면은 그 하나하나가 한국 사회와의 상호작용 속에서 진화해온 과정을 보여주는 자료이지만, 개별 사례마다 연구자의 정성적인 평가가 필요한 전통적인 방법론으로는 폭발적으로 늘어난 아파트에 대한 전수조사가 불가능하다. 때문에 많은 연구가 특정 지역이나 시기를 한정하여 수행되고, 다른 지역 또는 시기와 비교할 때는 선행연구의 방법론을 그대로 적용하게 된다. 이러한 방식으로는 기존 연구에서 설정된 틀을 벗어나기 어렵다. 기계학습은 한국 아파트 전체에 대한 분석을 가능하게 하고, 동시대에는 직관적으로 발견할 수 없었던 실체를 다양한 측면에서 탐색할 수 있게 한다.

## 1.2 연구의 목적

이 연구의 목적은 한국 아파트 전수에 대한 단위평면 분석이 가능한 방법론을 딥 러닝 방법론을 활용하여 개발하는 것이다. 이러한 방법론은 데이터에 잠재된 규칙을 찾아내는 기계학습의 방법론을 적용하여 귀납적인 연구를 수행할 수 있게 한다는 점에서, 선험적 기준에 따라 연역적으로 데이터를 분석하는 선행연구들과 차별화된다. 또한, 외국의 아파트와 다른 한국 아파트만의 공간을

기계학습을 이용하여 분석하는 첫 시도로서의 의의가 있다.

이를 위하여, 한국 아파트 단위평면에 대한 데이터셋을 구축하고, 그 안에 잠재된 규칙을 학습하고 해석 가능한 형태로 도출할 수 있는 딥 러닝 모형을 선정하고 적용한다. 개별 평면에 대한 수작업 없이 이러한 분석이 가능하도록, 데이터셋 구축부터 딥 러닝 기계학습 적용까지 분석의 모든 과정을 자동화한다.

이렇게 개발된 방법론을 한국 아파트의 진화과정 분석에 적용하여 검증한다. 한국 아파트가 시기별로 어떻게 변화하였는지는 이미 많은 선행연구의 주제이기도 하다. 따라서 딥 러닝 모형이 한국 아파트에 대한 전수조사를 통하여 발견한 한국 아파트의 시기별 변화를 확인하고, 그 해석을 한국 아파트에 대한 선행연구의 발견과 비교하여 분석 방법론을 검증하고 한국 아파트의 진화과정에 대한 새로운 시각을 도출한다.

이러한 과정을 통하여 기계학습과 딥 러닝을 적용한 건축 공간 연구의 가능성을 탐색하는 것이 이 연구의 궁극적 목표이다.

### 1.3 연구의 방법

이 연구는 기계학습 및 딥 러닝 방법론의 건축 분야 활용 분석, 한국 아파트 단위평면 분석 방법론과 데이터셋 구축, 딥 러닝을 활용한 한국 아파트 단위평면 분석, 주거공간 배치유형에서 나타나는 한국 아파트의 진화과정 분석의 절차로 이루어진다.

2장에서는 기계학습 및 딥 러닝 연구의 동향과 이를 건축 분야에 활용하는 선행연구를 고찰한다. 딥 러닝은 다양한 신경망 기계학습 모형과 관련 기법을 아우르는 개념이다. 이 중에서 어떠한 모형과 방법론이 건축 공간 관련 연구에 접목되어 활용되고 있는지 문헌 고찰을 통하여 살펴본다. 선행연구의 방법론

을 한국 아파트에 적용할 때 발생하는 한계를 파악하고, 이를 반영하여 한국 아파트 단위평면에 딥 러닝을 적용하기 위한 전략을 모색한다.

3장에서는 딥 러닝을 활용하여 한국 아파트 단위평면에서 나타나는 주거공간의 배치를 분석하는 방법론을 수립한다. 2장에서 고찰한 내용을 바탕으로, 주거공간 배치를 딥 러닝 모형에 입력할 수 있는 형태로 모형화한다. 딥 러닝 모형이 데이터에서 학습한 주거공간 배치 특성을 분석할 수 있는 방법론을 개발한다. 또한, 이 장에서 개발된 분석 방법론을 한국 아파트 단위평면 데이터셋에 적용하여 검증할 수 있는 방법론을 함께 수립한다.

4장에서는 한국 아파트 단위평면 데이터셋을 구축한다. 한국 아파트에 대한 전수조사를 딥 러닝 방법론을 통해 수행하기 위하여 필요한 단위평면 데이터셋을 구축한다. 공개된 단위평면 자료를 수집하고, 수집된 평면에서 주거공간의 배치 정보를 인식하여, CNN (convolutional neural network, 합성곱 신경망)의 입력값으로 적용할 수 있도록 3차원 행렬 형태로 처리한다. 이러한 과정 전체를 자동화하여 개별 단위평면에 대한 수작업 없이 전수조사가 가능하도록 하고, 데이터셋 구축 과정의 재현가능성을 확보한다.

5장에서는 한국 아파트 단위평면의 시기별 계획 특성을 3장에서 개발한 분석 및 검증 방법론과 4장에서 구축한 단위평면 데이터셋을 활용하여 분석한다. 한국 아파트 데이터셋에서 나타나는 준공연도와 주거공간 배치 계획 사이의 관계를 분석하고, 딥 러닝 모형이 학습한 주거공간 배치를 유형화하여 딥 러닝을 활용한 건축 공간 분석 방법론의 활용 가능성을 검증한다.

6장에서는 5장에서 도출된 주거공간 배치유형을 통하여 단위평면에서 나타나는 한국 아파트의 진화과정을 분석한다. 한국 아파트에서 전형적으로 나타나는 다양한 단위평면에 대하여, 5장의 딥 러닝 모형이 인식하는 배치 특성을 분석하고, 그 결과를 선행연구와 비교하여 건축적으로 해석한다. 이러한 과정을 통하여, 이 연구에서 개발한 분석 방법론을 건축 계획 연구에 적용할 수 있는

가능성을 검증한다.

## 제 2 장

# 건축 공간과 딥 러닝

이 장에서는 기계학습과 딥 러닝 방법론을 살펴보고 딥 러닝을 건축 분야에 활용하는 선행연구를 검토하여 건축 공간 연구에 대한 딥 러닝의 적용 가능성을 고찰한다. 이를 통하여 딥 러닝을 활용한 한국 아파트 단위평면 연구의 방법론적 요구조건을 확인한다.

### 2.1 기계학습

기계학습은 컴퓨터가 사람과 같은 지능적 판단을 할 수 있도록 하는 인공지능의 한 갈래이다. (그림 2.1) 기계학습을 다른 인공지능 방법론과 구분짓는 가장 큰 특징은 사람의 사고 방식을 닮는 것이 목표가 아니라는 점이다. 기계학습 모형은 데이터에서 귀납적으로 규칙을 학습하여 다양한 작업을 수행할 수 있다. 기계학습이 처리할 수 있는 작업에는 분류 (classification), 회귀분석 (regression), 새로운 사례의 합성 (synthesis), 결측값의 대체 (imputation of missing values) 등이 포함된다.<sup>10</sup> 이러한 접근 방식은 사람의 판단 과정을 컴퓨터에 그대로 옮기는 것이 목표인 전문가 시스템 (expert system)과는 정

---

<sup>10</sup>Goodfellow 등. (2016). Deep Learning. MIT Press., pp. 96–101

반대이다. 기계학습은 사람의 개입 없이 데이터에서 규칙을 스스로 도출하는 비지도 (unsupervised) 방법론이기 때문에 사람이 그 판단 과정을 설명하기 어려운 작업법도 학습할 수 있다는 장점이 있다.

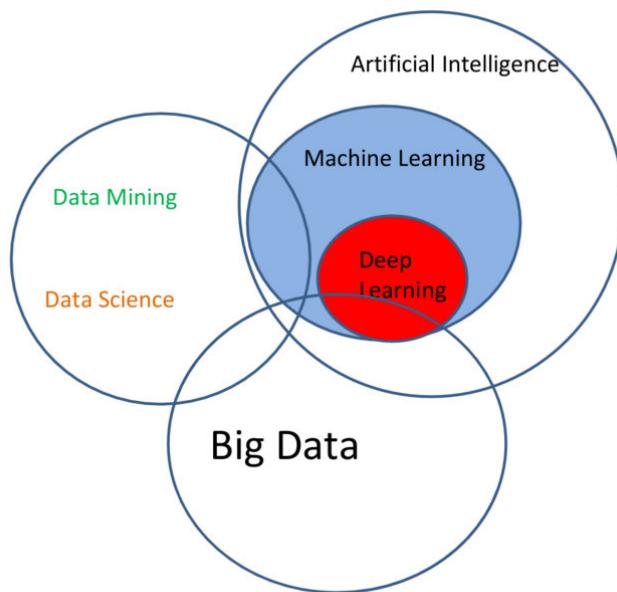


그림 2.1: 인공지능, 기계학습, 딥 러닝의 관계<sup>11</sup>

데이터 과학의 다양한 개념을 벤 다이어그램으로 표현한 그림 2.1에서 기계학습은 데이터에서 의미 있는 통계적 사실을 발견하는 데이터 마이닝 (data mining)과 전통적인 방식으로 처리하기 어려운 규모의 데이터를 다루는 빅데이터 (big data) 등의 개념과 교집합을 갖는다. 특히 데이터가 많고 폭넓을수록 기계학습의 귀납적 방법론을 통하여 더 유용한 규칙을 도출할 수 있기 때문에 모든 데이터에 대한 전수조사를 수행할 수 있는 빅데이터와 깊은 연관이 있다.

기계가 스스로 학습하게 하는 목적을 달성하기 위하여 다양한 구조와 학습 방식을 지닌 기계학습 모형들이 개발되었다. 이 모형들은 학습이 가능한 데이터와

---

<sup>11</sup>Piatetsky-Shapiro. (2016). Data science Venn diagram.

학습된 규칙의 특성 측면에서 매우 다양한 모습을 보인다. 그림 2.2은 빨간 점과 파란 점으로 표현된 데이터셋 사례들에 대하여 다양한 기계학습 모형들이 서로 다른 색의 점이 나타나는 영역을 구분짓는 규칙을 학습한 결과를 나타낸 것이다. 기계학습 모형에 따라 같은 데이터에서도 서로 다른 규칙이 도출되며, 학습된 규칙의 표현 능력과 제약 또한 크게 다른 것을 알 수 있다.

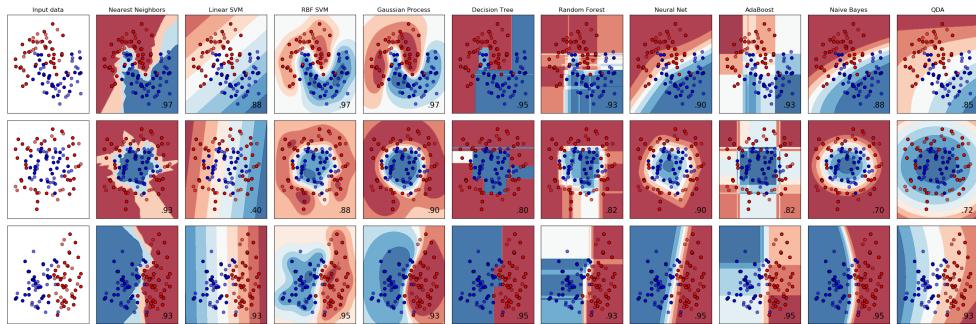


그림 2.2: 다양한 기계학습 모형 비교<sup>12</sup>

이렇게 다양한 기계학습 모형은 각각 다른 성격의 문제를 잘 해결할 수 있다. 그러나 사람은 매우 쉽게 처리하지만 컴퓨터로 처리하기에는 매우 어려운 과제도 존재한다. 사람은 사진을 보고 그 안의 사물이 무엇인지 매우 쉽게 인식할 수 있다. 그러나 컴퓨터는 사진을 수없이 많은 픽셀의 집합으로 입력받는다. 각 픽셀은 독립적인 값을 가질 수 있는 변수이고, 사진 한 장은 수백만 개의 요인을 고려해야 하는 문제가 된다. 차원의 저주 (curse of dimensionality)는 이렇게 데이터의 복잡성이 높아질수록 기계학습으로 문제를 해결하기가 지수함수적으로 어려워지는 현상을 의미한다.<sup>13</sup> 이처럼 사람이 일상적으로 처리하는 작업의 대부분은 여러 기계학습 모형을 적용할 수 없을 정도로 높은 복잡성을 지닌다. 그러나 이렇게 복잡도가 높은 과제를 해결할 수 있는 기계학습 방법론인 딥 러닝이 등장하면서 기계학습의 귀납적 접근 방식을 더욱 다양한 분야에 적용하게 되었다.

<sup>12</sup>Scikit-learn developers. (2019). Classifier comparison.

<sup>13</sup>Goodfellow 등. (2016). Deep Learning. MIT Press., pp. 152–154

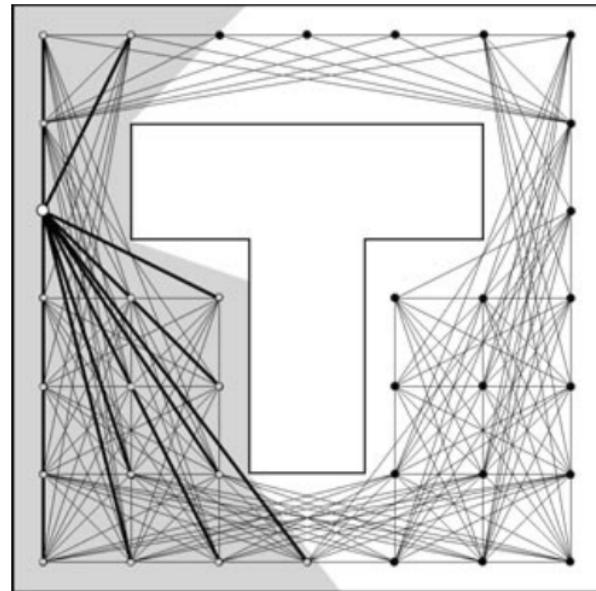
귀납적 방법론으로서의 기계학습의 성격을 마찬가지로 정량적 방법론인 공간구문론 (space syntax)과 비교하면 그 차이가 더 잘 드러난다. 공간구문론에서 사회 현상에 영향을 미치는 공간의 구조를 재현하는 방식은 단위공간과 그 연결 관계를 선형적인 기준에 따라 결정하는 것이다. 축선도 (axial map)와 볼록공간도 (convex map)는 같은 공간의 구조를 서로 다른 두 개의 그래프로 표현한다. 축선도에서 공간의 최소 단위인 단위공간은 일직선의 가시선인 축선 (axial line)이고, 서로 교차하는 축선을 연결한 그래프가 공간 구조로 도출된다. 반면, 볼록공간도에서는 모든 경계가 볼록하여 내부의 모든 지점이 서로 보이는 볼록공간 (convex space)이 단위공간이 된다. 축선도에서 한 지점이 여러 축선에 속하는 교차점이 될 수 있었던 것과 달리, 볼록공간도는 각 단위공간이 겹치지 않도록 작성된다. 이렇게 두 방법론은 공간 구조를 전혀 다른 단위공간과 그 연결 그래프로 재현하지만, 가시성 (visibility)이라는 선형적 기준에서 연역적으로 도출되었다는 점은 같다. 이러한 특성은 다른 공간분석 모형에도 동일하게 적용된다.

공간분석 모형 중 하나인 가시성 그래프 (visibility graph)는 공간을 2차원 격자로 분할하여 격자점 하나하나를 단위공간으로 삼는다.<sup>14</sup> 따라서 축선도나 볼록공간도와 달리 단위공간의 분할에 작성하는 사람의 주관에 따라 달라질 수 있는 여지가 없다. 이러한 설정은 기계학습 모형이 컴퓨터 이미지를 분석하는 경우와 매우 유사하다. 그러나 단위공간의 연결 관계를 상호 가시성에서 도출하고, 공간구문론의 그래프 중심성 지표를 적용하여 사회 현상과 비교하는 등을 연구자가 선형적으로 결정하는 연역적 방법론이라는 점은 가시성 그래프 분석에서도 다르지 않다. (그림 2.3가)

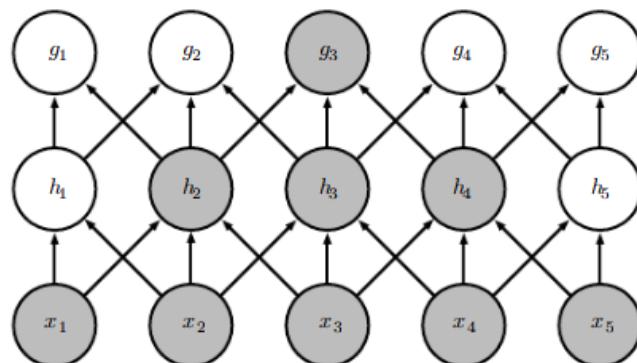
<sup>14</sup>Turner 등. (2001). From Isovists to Visibility Graphs: A Methodology for the Analysis of Architectural Space. Environment and Planning B: Planning and Design, 28(1), 103-121.

<sup>15</sup>Turner 등. (2001). From Isovists to Visibility Graphs: A Methodology for the Analysis of Architectural Space.

<sup>16</sup>Goodfellow 등. (2016). Deep Learning.



(가) 가시성 그래프<sup>15</sup>



(나) 딥 러닝의 계층간 연결<sup>16</sup>

그림 2.3: 가시성 그래프와 딥 러닝 모형 비교

기계학습에서는 반대로 주어진 데이터를 가장 잘 설명하는 규칙을 귀납적으로 도출한다. 기계학습 모형에서 각 입력값 사이에 존재하는 관계는 존재할 수 있는 모든 관계를 포괄하는 가능성을 표현한다. 기계학습의 핵심인 학습 알고리즘은 데이터에 내재한 규칙을 반영하여 그러한 관계의 강도를 조절하면서 데이터를 잘 설명하는 일부 관계만 남긴 모형을 귀납적으로 만들어낸다. 기계학습에서 입력값의 인식 및 처리를 위한 지표는 이러한 과정을 통하여 귀납적으로 도출된다. 따라서 하나의 기계학습 방법론으로 다양한 데이터를 해석하고 처리하는 모형을 얻는 것이 가능하다. (그림 2.3나)

## 2.2 딥 러닝

딥 러닝 (deep Learning)은 기계학습 모형 중 하나인 인공신경망 (artificial neural network)을 더욱 깊은 구조로 발전시킨 심층 신경망 (deep neural network)을 활용하는 기계학습 방법론이다. 생명체에서 뉴런은 전기 자극을 받으면 연결된 다른 뉴런들에 그 자극을 전달하는 단순한 구조를 지니고 있지만 수많은 뉴런이 서로 연결되어 뇌를 이루고 지능을 갖추듯이, 인공신경망도 단순한 계산을 수행하는 노드가 서로 연결되어 이어지는 신경망 구조를 차용한다. 입력층과 출력층 사이에 여러 은닉층을 갖춘 심층 신경망은 여러 계층 구조를 이용하여 단순한 개념이 결합된 복잡한 개념을 학습하는 것이 가능하다. 그림 2.4은 심층 신경망 모형이 물체의 모양을 인식하는 과정을 보여준다. 단순한 경계선의 인식에서 시작해 점점 더 복잡한 모양을 인식하는 계층을 거쳐 사람과 자동차 등의 특성을 식별할 수 있게 된다.<sup>17</sup>

CNN (convolutional neural network, 합성곱 신경망)은 픽셀로 이루어진

---

<sup>17</sup>Goodfellow 등. (2016). Deep Learning., pp. 5–6

<sup>18</sup>Goodfellow 등. (2016). Deep Learning., p. 6; Zeiler, Fergus. (2014). Visualizing and understanding convolutional networks. In European conference on computer vision (pp 818–833). Springer. 에서 재구성

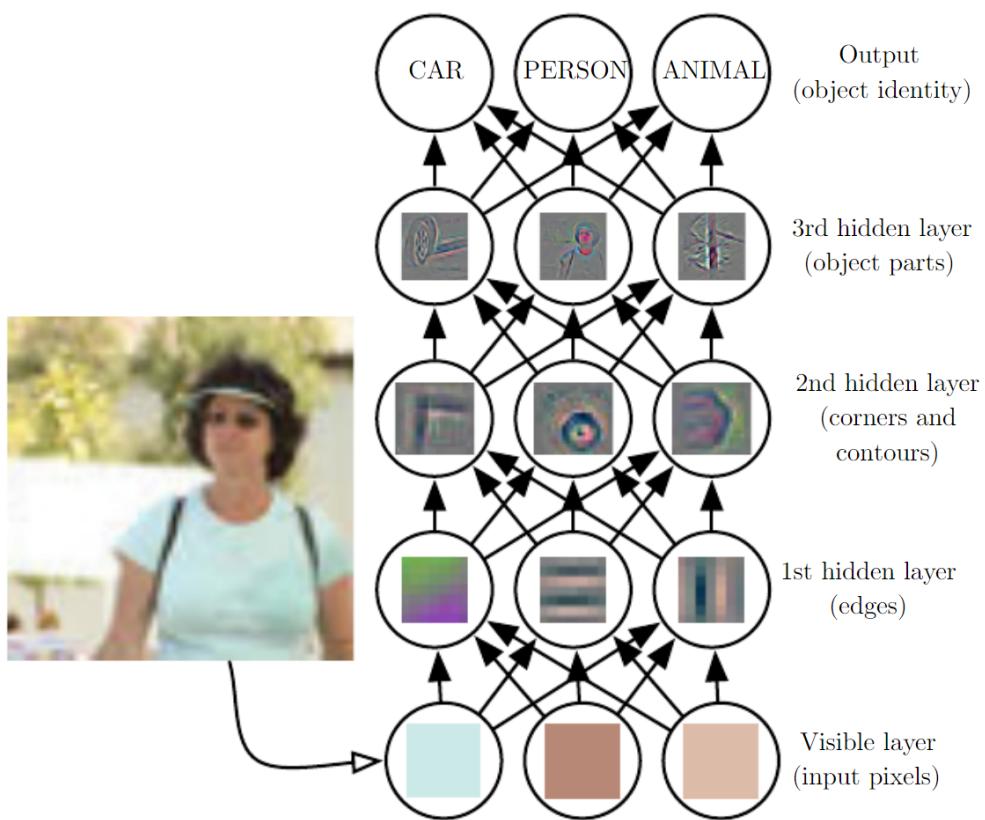


그림 2.4: 심층 신경망 모형의 구조<sup>18</sup>

컴퓨터 이미지와 같이 격자 구조로 구성된 데이터를 처리하는 데 특화된 딥 러닝 모형이다.<sup>19</sup> 바둑판이나 건축 공간을 모형화한 가시성 그래프 등 일정 간격의 격자 구조로 표현할 수 있는 모든 대상은 CNN의 입력값이 될 수 있다. 바둑 인공지능인 알파고에도 착수 위치를 분석하는 정책망과 승률을 예측하는 가치망에 CNN이 적용되었다. (그림 2.5)

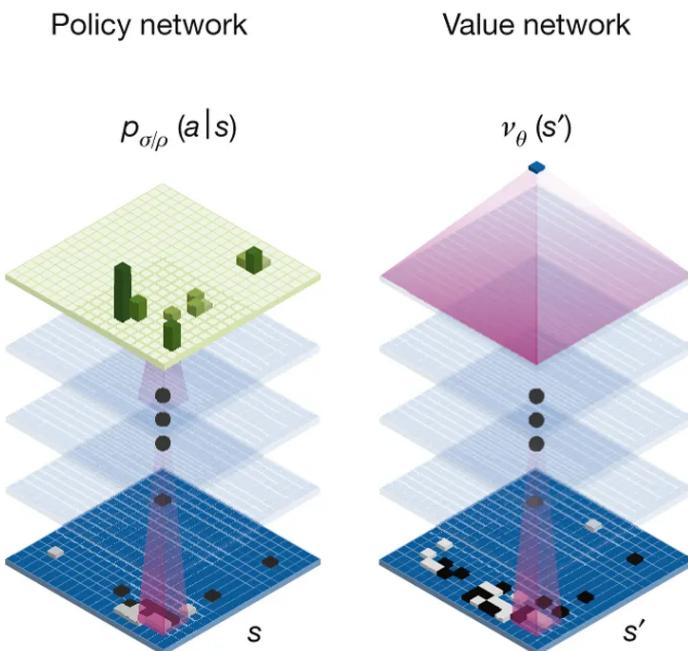


그림 2.5: 알파고에 사용된 합성곱 신경망<sup>20</sup>

딥 러닝은 더 많은 노드를 더 깊은 구조로 연결한 신경망을 통하여 더 어려운 과제를 수행할 수 있다. CNN은 더 깊은 신경망 구조를 달성하기 위하여 격자 구조에서 나타나는 공간적 관계를 반영하여 공간적으로 서로 인접한 노드만을 연결하는 방식으로 모형의 복잡도를 낮춘다. CNN은 이러한 구조를 통하여 더 깊은 구조의 신경망을 학습시킬 수 있고, 영상 인식부터 바둑 승률 예측까지

<sup>19</sup>Goodfellow 등. (2016). Deep Learning., p. 326

<sup>20</sup>Silver 등. (2016). Mastering the game of Go with deep neural networks and tree search. Nature, 529(7587), 484–489.

데이터의 공간적 관계가 중요한 과제를 잘 처리할 수 있다.

딥 러닝은 다른 기계학습 모형과 비교할 때 동일한 수준의 모형을 학습시키기 위하여 더 많은 데이터와 더 많은 계산을 필요로 한다. 비교적 단순한 문제에도 깊은 구조의 모형이 필요한 것은 오래 전부터 연구되었던 기계학습 방법론인 신경망이 최근에서야 두각을 나타내는 이유이기도 하다. 시간이 지날수록 점점 더 많은 데이터에 대한 접근이 가능해지면서 딥 러닝 학습을 위한 데이터셋의 규모도 커지고, 이를 활용하여 더 어렵고 복잡한 과제를 처리할 수 있는 더 크고 깊은 구조의 신경망이 딥 러닝에 활용되고 있다. 손으로 쓴 0부터 9까지 10가지 숫자를 구분하는 MNIST 데이터셋은 수만 개의 손글씨 이미지를 담고 있으며, 많은 기계학습 모형이 이를 활용하여 사람과 같은 수준으로 손글씨를 구분할 수 있다. 그러나 사진 속에서 약 1만 가지 종류의 대상을 분류하는 것과 같은 더 어려운 과제에는 더 많은 데이터가 필요하기 때문에, 이를 위한 ImageNet10K 데이터셋은 약 1천만 개에 달하는 이미지를 포함한다. (그림 2.6가) 딥 러닝은 이렇게 큰 데이터셋을 활용하여 다른 기계학습 모형의 한계를 넘어선 더 어려운 과제를 수행할 수 있다.

4장에서 구축한 한국 아파트 단위평면 데이터셋의 경우 약 5만 개의 단위평면을 포함하고 있다. 유사한 규모의 데이터셋에서 나타나는 과제의 성격을 검토하면, 이 데이터셋에 딥 러닝을 활용하여 달성할 수 있는 과제의 수준을 예측할 수 있다. 사례 수가 수만 개 수준에 달하는 데이터셋은 10가지 숫자 손글씨를 구분하는 MNIST와 10가지 사진 속 사물을 구분하는 CIFAR-10 등과 유사한 규모이다. (그림 2.6가) 따라서 단위평면 데이터셋에 딥 러닝을 적용하여 분석하였을 때, 한국 아파트의 단위평면을 약 10가지 유형으로 분류하는 정도의 일을 처리할 것으로 기대할 수 있다.

컴퓨터의 성능이 발전하면서 딥 러닝의 학습 능력의 한계를 결정하는 신경망의 규모도 지수함수적 속도로 증가하고 있다. 이러한 성장은 현재 벌과 개구리의 뇌를 이루는 뉴런의 수와 비슷한 규모의 딥 러닝 모형이 21세기 중반에는 인

간의 뇌에 맞먹는 크기에 도달할 것이라는 예측으로 이어진다. (그림 2.6나) 컴퓨터의 성능이 계속 발전하고 딥 러닝 연구가 진전되어 이러한 추세가 계속 이어진다면, 앞으로도 다양한 분야에서 딥 러닝이 인간만이 할 수 있었던 일을 딥 러닝을 통해 처리하게 될 것이다.

### 2.3 건축 공간에 대한 딥 러닝

기계학습과 딥 러닝은 사람의 개입 없이 데이터에 내재된 규칙을 발견하고 그에 따른 작업을 자동으로 수행할 수 있기 때문에 건축을 포함한 다양한 분야에서 활용 가능성이 높다. 특히 CNN은 2차원 이미지에 대한 분석에서 매우 뛰어난 성과를 보여주고 있으므로 평면도 이미지나 가시성 그래프처럼 건축 공간을 2-3차원 격자점으로 재현하여 분석하는 연구에 적합하다. 따라서 건축 분야에서 딥 러닝을 활용한 연구가 이미 많이 이루어지고 있다.

아파트의 단위평면에 딥 러닝을 적용한 연구는 ArchiGAN이 있다. ArchiGAN은 보스턴 지역의 아파트 계획을 딥 러닝 모형에 학습시켜, 대지 조건에서 주동 배치, 주동 내 단위세대 배치, 단위세대 내 주거공간 배치, 실내 가구 배치를 연쇄적으로 생성하는 모형이다.<sup>22</sup>

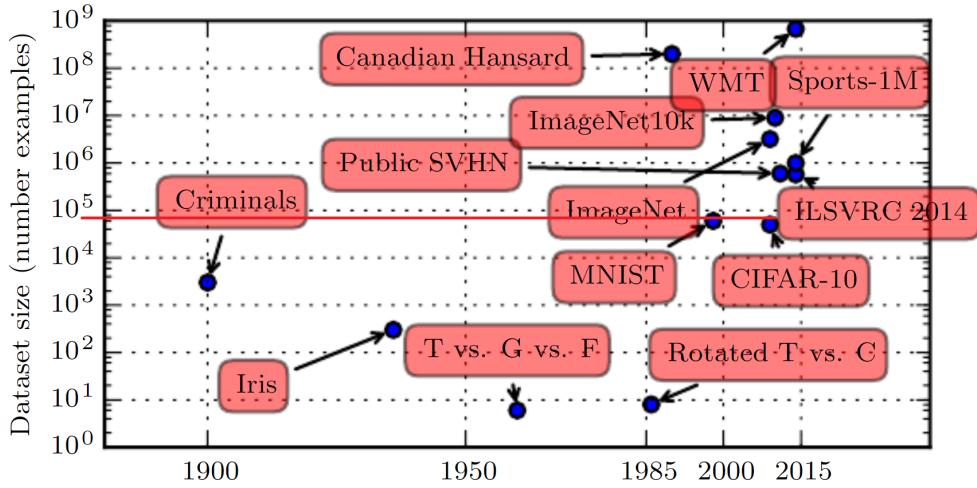
ArchiGAN은 이미지를 변환하는 딥 러닝 모형인 Pix2Pix를 활용하였다.<sup>23</sup> Pix2Pix는 서로 대응되는 이미지 사이의 관계를 학습할 수 있으며, 학습하는 데이터에 따라 다양한 활용이 가능하다. 그림 2.7는 Pix2Pix가 대상의 종류와 위치를 표현한 이미지에서 도로주행과 건물 입면 사진을 생성하고, 윤곽선에서 가방의 사진을 그려내는 등 주어진 데이터에 따라 다양한 종류의 작업을 수행할 수 있다는 것을 보여준다.<sup>24</sup>

<sup>21</sup>Goodfellow 등. (2016). Deep Learning. MIT Press., pp. 19, 23

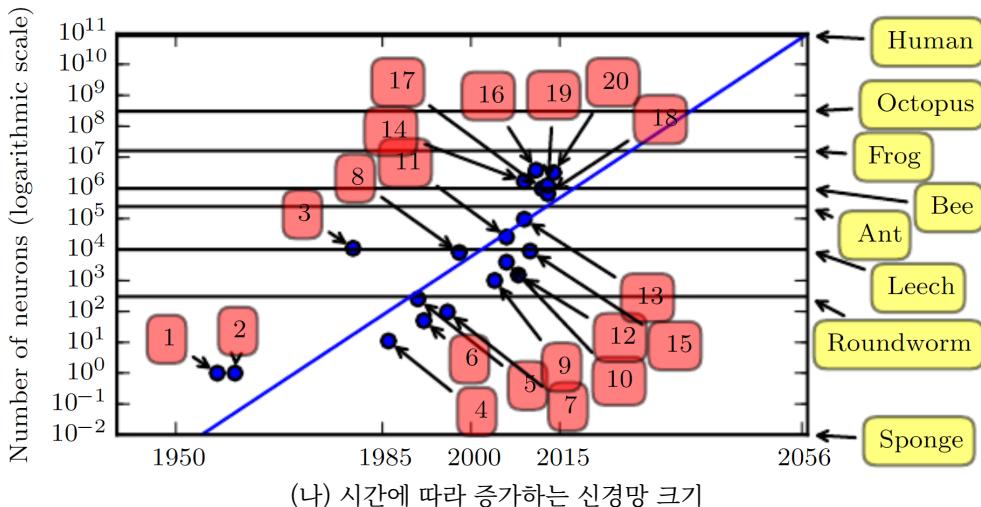
<sup>22</sup>Chaillou. (2019). AI + Architecture (mathesis). Harvard University.

<sup>23</sup>Chaillou. (2019). AI + Architecture.

<sup>24</sup>Isola 등. (2016). Image-to-Image Translation with Conditional Adversarial Networks. arxiv.



(가) 시간에 따라 증가하는 데이터셋 크기 (빨간 선은 4장의 한국 아파트 데이터셋의 크기)



(나) 시간에 따라 증가하는 신경망 크기

그림 2.6: 딥 러닝의 성장 속도<sup>21</sup>

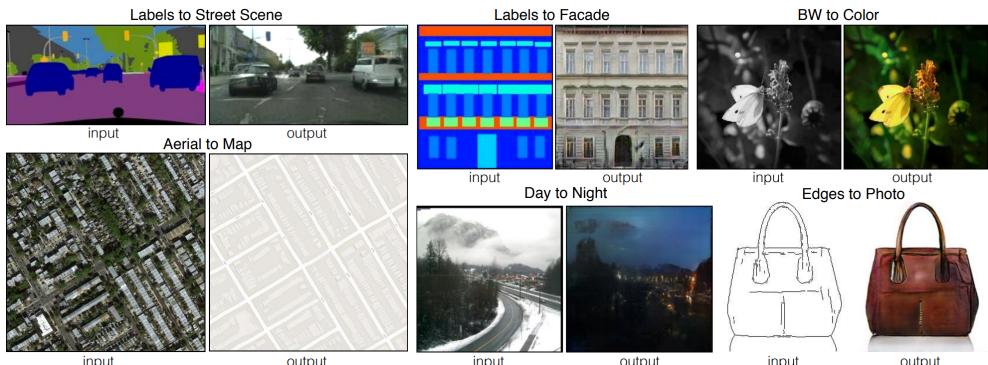


그림 2.7: Pix2Pix<sup>25</sup>

ArchiGAN은 대지 조건에서 아파트 평면을 생성하는 작업을 4단계의 이미지 변환 작업으로 분할하여, 그 중 간단한 알고리즘으로 수행할 수 있는 주동 평면 분할을 제외한 주동 배치, 단위세대 내 주거공간 배치, 실내 가구 배치 3단계에 Pix2Pix를 적용하고, 수작업으로 구축한 보스턴 아파트 평면 데이터로 학습시켜 만들어진 모형이다.<sup>26</sup>

Pix2Pix와 이에 기반한 ArchiGAN은 GAN (generative adversarial network, 적대적 생성 신경망) 구조로 이루어져 있다. GAN은 훈련용으로 주어진 실제 데이터와 가짜로 생성된 것을 구분하는 판별자 (discriminator)와 판별자를 속일 수 있는 데이터를 생성하는 생성자로 이루어진다. 창과 방패의 싸움처럼, 판별자와 생성자는 서로 적대적으로 속이고 속지 않기 위한 경쟁을 반복한다. 이러한 과정을 통하여 생성자는 실제 데이터와 분간하기 어려운 가상의 데이터를 생성할 수 있게 된다.

ArchiGAN에서 단위세대 내 주거공간 배치를 생성하는 단계는 보스턴 아파트 단위평면의 주거공간을 거실, 침실, 화장실, 복도, 주방, 벽장 6가지로 분류한 배치를 이미지 형태의 훈련 데이터로 활용한다. 이러한 데이터를 활용하여 GAN

<sup>25</sup>Isola 등. (2016). Image-to-Image Translation with Conditional Adversarial Networks.

<sup>26</sup>Chaillou. (2019). AI + Architecture.

은 실제 평면의 주거공간 배치와 가짜를 구분하는 판별자 (discriminator) 와 판별자를 속일 수 있는 가상의 평면을 만들어내는 생성자 (generator) 두 신경망을 서로 적대적으로 학습시킨다. 이러한 학습 과정을 통하여 생성자가 사람이 보기에 실제 보스턴의 아파트와 구분할 수 없는 새로운 평면을 생성하도록 한다. 그림 2.8은 다양한 단위세대 형태에 대하여 왼쪽의 외부조건을 입력하면 가운데와 같은 배치를 출력하는 것을 보여준다. 이를 오른쪽의 실제 보스턴 아파트의 주거공간 배치와 비교하면 약간씩 다른 점을 발견할 수 있다. 이를 통하여 ArchiGAN이 단순히 훈련 데이터로 주어진 배치를 기억하는 것이 아니라 여러 평면에서 공통적으로 나타나는 배치 계획의 성격을 학습하였다는 점을 확인할 수 있다.<sup>27</sup>

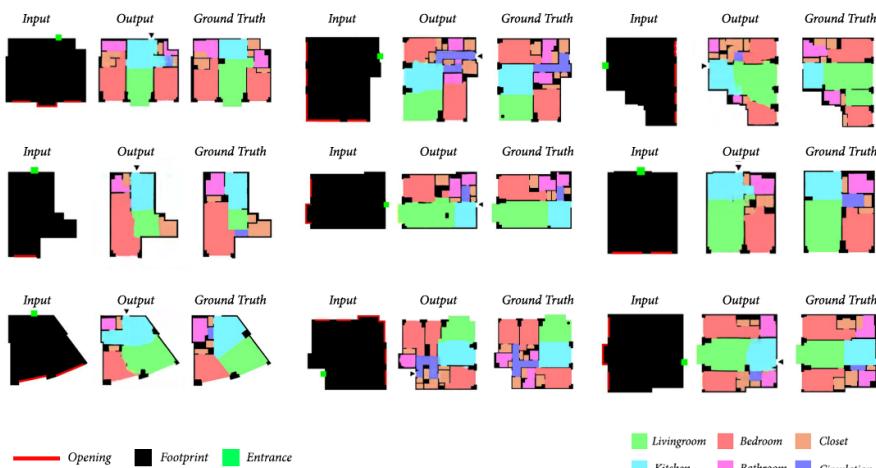


그림 2.8: ArchiGAN<sup>28</sup>

ArchiGAN이 보스턴 아파트의 단위평면을 학습하여 실제와 흡사한 평면을 생성할 수 있는 것이 모든 조건에서 아파트의 단위평면을 설계할 수 있는 능력을 갖추었다는 뜻은 아니다. 기계학습은 귀납적으로 데이터에서 잠재된

<sup>27</sup> Chaillou. (2019). AI + Architecture.

<sup>28</sup> Chaillou. (2019). AI + Architecture., p. 42

규칙을 찾아내는 방식이기 때문에, 데이터가 포괄하는 범위를 벗어난 외삽은 가능하지 않다. 보스턴 아파트만으로 학습한 딥 러닝 모형이 다른 시공간적 환경에서 아파트 평면을 계획할 수는 없다는 것이다. 실제로 ArchiGAN에 한국 아파트에서 전형적인 계단실형 단위세대에 적용하면 그림 2.9와 같이 주거의 기본 조건을 갖추지 못한 계획을 도출한다. 전면 전체와 후면의 절반이 침실로 채워지고, 화장실과 복도여야 할 공간은 방의 형태를 이루지 못하고 흩어져 있는 배치를 보인다. 이러한 결과는 당연히 보스턴 아파트의 평면과 다르지만, 전면에서 후면까지 이어진 통합된 LDK를 중심으로 각 실이 배치된 한국 아파트의 주거공간 배치와도 전혀 다른 모습이다.

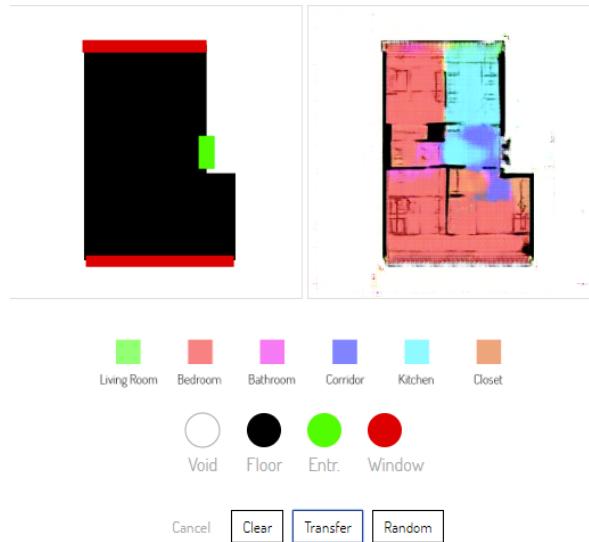


그림 2.9: ArchiGAN을 한국 아파트 단위세대에 적용한 결과<sup>29</sup>

이와 같은 사례를 통하여 2차원 이미지로 재현된 건축 평면에 대한 분석과 재현에 딥 러닝을 활용할 수 있음을 확인하였다. 그러나 딥 러닝을 포함한 기계학습은 주어진 데이터 안에서만 규칙을 발견할 수 있다는 한계가 있다. 보스턴 아파트를 대상으로 학습한 딥 러닝 모형으로는 현재까지 지어진 보스턴 아파트를 재현하는 것에만 적용할 수 있으며, 다른 지역의 아파트나 미래에

<sup>29</sup>Chaillou. (2019). AI + Architecture.

생겨날 새로운 조건의 아파트를 계획하는 능력을 갖출 수 없다. 따라서 한국 아파트를 분석하기 위해서는 한국 아파트에 대한 데이터로 학습한 딥 러닝 모형이 필요하다는 것을 알 수 있다.

## 2.4 딥 러닝의 해석

기계학습 모형은 사람의 개입과 지도 없이도 데이터 안에 내재된 규칙을 귀납적으로 발견할 수 있다. 그러나 기계학습으로 학습된 규칙은 사람이 이해하기 어려운 경우가 많다. 특히 딥 러닝은 깊은 신경망 구조에서 나오는 높은 복잡도를 활용하여 다양한 분야에서 다른 기계학습 방법론보다 뛰어난 성능을 보이지만, 그 복잡성에 비례하여 학습된 규칙을 사람이 이해하는 것이 블랙박스에 비유될 정도로 어렵다.

기계학습을 통하여 학습된 규칙이 사람이 기대한 것과 완전히 다른 경우도 많다. 이러한 특성은 입력값을 약간만 조작하여 전혀 다른 판단을 이끌어내는 적대적 공격 (adversarial attack) 에서 잘 드러난다. 사람은 정지 표지판이 어떤 모양과, 색, 글자로 이루어져 있는지로 인식하고, 많은 부분이 생략되거나 상당한 수준으로 변형되어도 알아볼 수 있다. 그러나 여러 교통 표지판을 잘 구분할 수 있는 딥 러닝 모형도 그 모형의 판단 기준에 대한 맞춤형 공격에 취약하다. 예를 들어, 정지 표지판에 스티커 몇 조각을 붙이기만 하여도 전혀 다른 형태의 속도제한 표지판으로 인식할 수 있다 그림 2.10나.<sup>30</sup> 이러한 공격이 가능한 이유는 사람이 성장 과정에서 학습한 모양, 색, 글자 등의 개념을 여러 작업에 공통적으로 적용하는 것과 달리, 기계학습 모형은 데이터셋 안의 사례를 구분하기 위한 최소한의 판단 기준만을 학습할 수 있기 때문이다. 교통 표지판을 잘 분류하는 딥 러닝 모형이 어떤 기준으로 그 작업을 수행하는지 알 수 없다는 점은 궁극적으로 사람이 인공지능을 신뢰할 수 있는지에 대한

---

<sup>30</sup>Eykholt 등. (2017). Robust Physical-World Attacks on Deep Learning Models.

문제로도 이어진다.

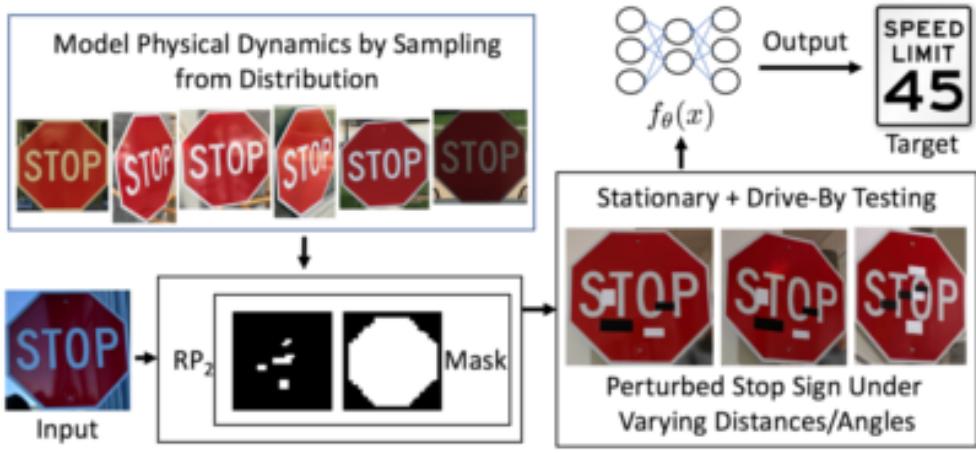
이처럼 딥 러닝이 학습한 규칙을 파악하기 어려울 뿐만 아니라 사람의 사고 방식에 비추어 짐작하기도 불가능하다는 점은 건축 등 다른 분야의 연구에 활용하고자 할 때 더 문제가 된다. 작동 원리를 알 수 없어도 사람 대신 자동으로 작업을 수행할 수만 있으면 되는 것이 아니라, 기계학습을 적용하여 데이터에서 규칙을 발견하는 것 자체가 목적이기 때문이다. 딥 러닝의 이러한 단점을 극복하기 위하여, 딥 러닝 모형이 데이터에서 학습한 규칙을 사람이 이해할 수 있도록 설명하기 위한 기술도 연구된다.

잠재공간 (latent space) 분석은 딥 러닝 모형이 데이터에서 학습한 규칙의 성격을 확인할 수 있는 방법론이다. 딥 러닝 모형에서 입력 데이터를 인식한 결과는 은닉층의 노드들이 가지는 값으로 표현된다. 입력값 전반에 걸친 특성을 인식한 결과는 흔히 출력층 직전 계층의 값을 통해 파악할 수 있다. 이 때 노드 하나하나의 값을 한 차원 상의 좌표로 해석하면, 입력 데이터를 처리한 후에 해당 계층의 노드들이 갖게 되는 값은 다차원 벡터공간 위의 한 점의 좌표에 대응되는데, 이 다차원 벡터공간이 잠재공간이다. 물론 일반적인 딥 러닝 모형에서 한 계층에는 수백 수천 개 이상의 노드가 존재하므로 잠재공간도 사람이 인식할 수 없는 고차원 공간이 된다. 따라서 사람이 인식할 수 있도록 주성분 분석 (principal component analysis, PCA)이나 독립 성분 분석 (independent component analysis, ICA) 등을 통한 차원 축소 (dimensionality reduction)를 고차원 잠재공간에 적용하여 2-3차원 공간에 시각화한다.

그림 2.11은 회화 작품의 이미지를 20가지 미술 양식으로 분류하는 딥 러닝 모형의 잠재공간을 차원축소를 통하여 3차원 공간에 표현한 것이다. 잠재공간 안에서 딥 러닝이 인식한 미술 양식은 원점에서의 방향으로 구분된다. 같은 미술 양식에 속하는 여러 작품은 원점에서 한 방향으로 뻗어나가는 군집을

---

<sup>31</sup>Eykholt 등. (2017). Robust Physical-World Attacks on Deep Learning Models.



(가) 적대적 사례 생성 과정



(나) 속도제한 표지판으로 오인식되도록 변형한 정지 표지판

그림 2.10: 교통 표지판 인식 딥 러닝 모형에 대한 적대적 공격<sup>31</sup>

이룬다. 이렇게 잠재공간 분석을 통하여 딥 러닝이 서로 다른 미술 양식에서 나타나는 차이를 구분하여 인식하는 것을 확인할 수 있다.<sup>32</sup>

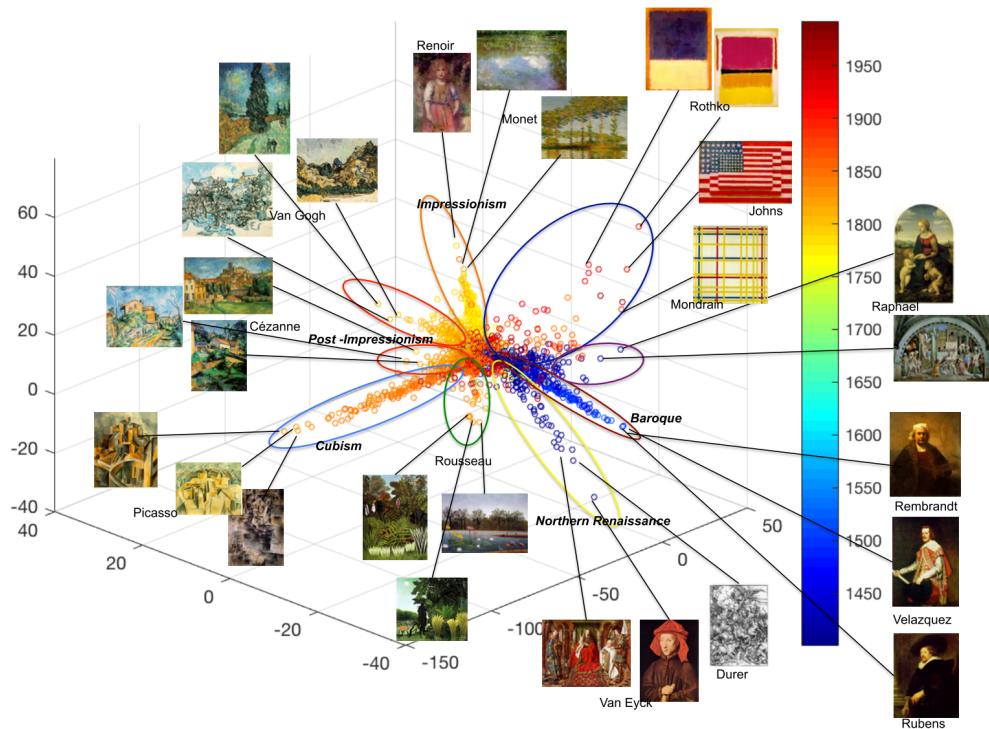


그림 2.11: 잠재공간 안에 재현된 미술 양식<sup>33</sup>

잠재공간과 실제 입력 데이터 사이의 관계는 비선형적이기 때문에, 딥 러닝 모형 내부에서 잠재공간을 구성하는 각 요인이 구체적으로 입력 데이터의 어떤 특성과 연관되는지를 파악하기는 어렵다. 잠재공간 분석을 통하여 이미지를 분류하는 딥 러닝 모형에서 특정 분류와 연관된 요인을 도출할 수는 있지만, 그렇게 분류된 이유를 알 수 없다는 문제가 그 이유가 되는 요인의 성격을 알 수 없다는 것으로 짚겨갈 뿐이다. 따라서 딥 러닝이 입력 데이터를 인식하는 과정에 대한 방법론 또한 연구된다.

<sup>32</sup>Elgammal 등. (2018). The Shape of Art History in the Eyes of the Machine.

<sup>33</sup>Elgammal 등. (2018). The Shape of Art History in the Eyes of the Machine.

역합성곱 신경망 (deconvolutional network)은 CNN 모형 내부의 합성곱 신경망이 입력 데이터에서 어떠한 패턴을 인식하는지를 시각화하는 딥 러닝 모형이다. 역합성곱 신경망은 CNN 모형의 각 계층을 가장 많이 활성화하는 입력값을 찾는다. 이를 통해 딥 러닝 모형이 인식한 패턴을 과장하여 강조하는 시각화를 얻는다. 그림 2.12에서 왼쪽 이미지에 나타나는 패턴은 오른쪽 이미지를 분류하기 위하여 학습된 딥 러닝 모형에서 그 이미지에서 인식된 패턴을 역합성곱 신경망을 통하여 재구성한 것이다. 9개 이미지로 이루어진 정사각형들은 잠재공간을 구성하는 하나의 요인에 대응한다. 임의로 선택된 10개 요인에 대하여, 해당 요인에 가장 잘 들어맞아 가장 강하게 활성화하는 9개 사례에서 이미지의 어떤 특징이 그 요인에 대응되는지를 시각화한다. 예를 들어, 윗줄 오른쪽의 경우 입력 이미지에는 다양한 대상이 나타나고 있지만, 해당 요인은 그 안에 있는 잔디밭에 집중하여 인식하고 있다는 것을 확인할 수 있다.<sup>34</sup>

위에서 살펴본 다양한 방법론을 활용하여 딥 러닝 모형의 내부 구조 파악이 어려운 단점을 극복하고, 딥 러닝이 데이터에서 학습한 규칙을 파악할 수 있다. 이를 통하여 데이터를 분석하여 내재된 규칙을 발견하는 것이 목적인 연구에서도 딥 러닝을 적용하는 것이 가능하다.

## 2.5 소결

이 장에서는 대규모 데이터에 대한 전수조사를 가능하게 하는 기계학습과 딥 러닝 방법론을 살펴보고 건축 공간에 적용하기 위한 전략을 고찰하였다. 기계학습은 데이터에서 스스로 규칙을 발견하는 귀납적 방법론이다. 기계학습 모형은 사람의 지도 없이도 주어진 데이터에 따라 다양한 작업법을 학습하고

---

<sup>34</sup>Zeiler, Fergus. (2014). Visualizing and understanding convolutional networks. In European conference on computer vision (pp 818–833). Springer.

<sup>35</sup>Zeiler, Fergus. (2014). Visualizing and understanding convolutional networks.

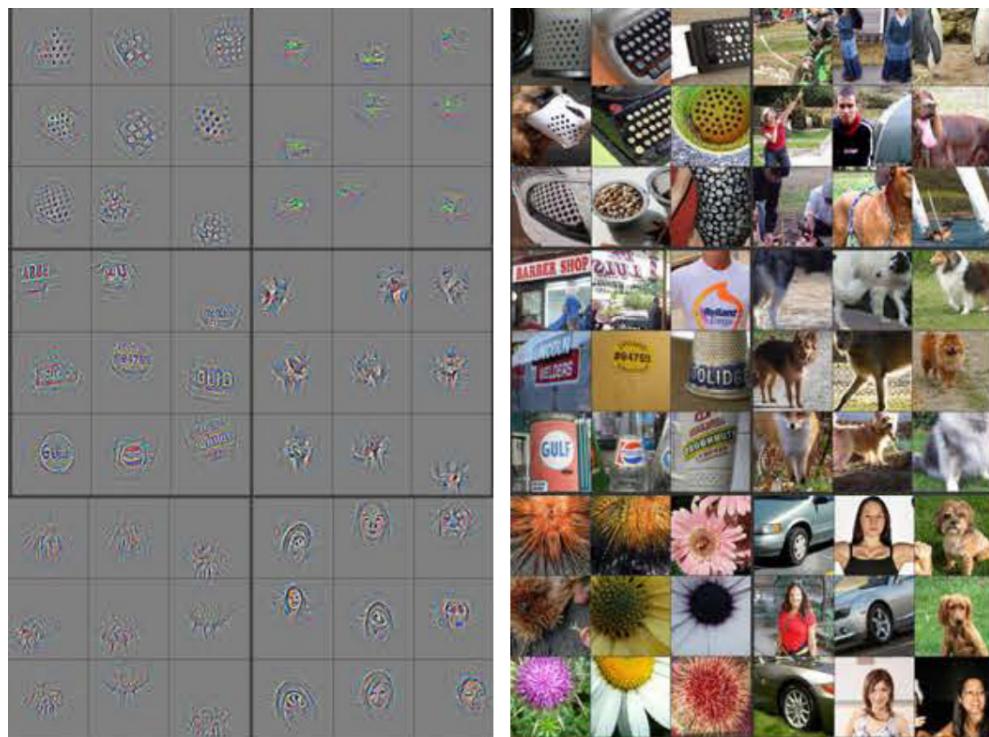


그림 2.12: 역합성곱 신경망으로 시각화한 딥 러닝의 이미지 인식<sup>35</sup>

그 작업을 수행할 수 있다. 따라서, 건축 공간에 대한 데이터를 기계학습 모형에 입력할 수 있는 형태로 구축하면 기계학습을 적용하여 건축 공간에 대한 분석이 가능하다.

딥 러닝은 단순한 구조로 다른 기계학습 모형보다 어렵고 복잡한 과제를 학습할 수 있다. 그러나 딥 러닝 모형을 학습시키는 데는 다른 기계학습 모형에 비해 서도 더 많은 데이터가 요구된다. CNN (convolutional neural network)은 격자 구조의 데이터에 적합한 딥 러닝 모형으로 이미지 인식을 포함한 다양한 분야에 활용되고 있다. 건축 공간 연구에 CNN을 적용하기 위해서는 가시성 그래프와 같이 건축 공간을 격자 구조로 재현하여야 하며, 매우 많은 수의 폭넓은 데이터가 필요하다.

건축에서도 기계학습과 딥 러닝은 이미 많은 연구에서 활용되고 있다. ArchiGAN은 GAN 모형 중 하나인 Pix2Pix를 활용하여 보스턴 아파트의 평면을 학습하여 대지 조건에서 평면을 생성한다. ArchiGAN은 아파트 평면 계획을 여러 단계의 이미지 변환 작업으로 분할하여 각 단계에 딥 러닝을 적용한다. 그러나 이 모형을 한국 아파트의 단위세대에 적용하면 한국 아파트 또는 보스턴 아파트와 유사한 평면을 도출하지 못하고 주거의 기본 조건을 갖추지 못한 배치를 내놓는다. 이러한 결과는 보스턴 아파트로 학습한 딥 러닝 모형을 한국 아파트 등 다른 조건에 활용할 수 없음을 보여준다. 따라서 한국 아파트에 딥 러닝을 적용하기 위해서는 한국 아파트에 대한 데이터를 구축하여 학습시킬 필요가 있음을 확인하였다.

기계학습이 데이터에서 귀납적으로 학습한 규칙은 사람의 예측과 크게 다를 수 있으며, 기계학습, 특히 딥 러닝이 학습한 규칙을 파악하기는 매우 어렵다. 딥 러닝의 내부 구조를 분석하여 딥 러닝이 데이터에서 학습한 규칙을 설명하기 위한 다양한 방법론이 있다. 잠재공간 분석은 딥 러닝 모형 안의 은닉층에 주성분 분석이나 독립 성분 분석 등의 차원 축소 기법을 적용하여 인식 결과를 설명하는 방법론이다. 잠재공간 속 데이터의 분포를 검토하여 딥 러닝이 데이

터에서 학습한 규칙을 확인할 수 있다. 그러나 잠재공간 분석은 입력 데이터에 대한 인식 결과를 설명할 수는 있어도, 데이터의 어떤 특성 때문에 그런 결과가 도출되었는지는 설명할 수 없다. CNN 모형에 대하여 입력 데이터와 인식 결과 사이의 관계를 간접적으로 확인하기 위한 방법론이 있다. 역합성곱 신경망은 CNN의 합성곱 신경망을 가장 활성화하는 입력값을 찾아 딥 러닝 모형이 인식하는 패턴을 시각화하는 방법론이다. CAM (class activation mapping)은 입력 데이터에서 인식 결과와 연관된 영역을 찾아 시각화하는 방법론이다. 이러한 방법론은 딥 러닝의 내부 구조 파악이 어려운 단점을 극복하고 데이터의 분석이 목적인 연구에도 딥 러닝을 적용할 수 있도록 한다.

이러한 고찰을 통하여, 한국 아파트에 대한 전수조사를 딥 러닝을 활용하여 수행하기 위하여 필요한 방법론적 요구조건을 확인하였다. 이미 건축 공간에 딥 러닝을 적용한 선행연구가 있으나, 한국 아파트에 대한 연구를 위해서는 한국 아파트 데이터셋이 필요하다. 한국 아파트의 공간을 2차원 격자 구조의 이미지로 재현한 데이터가 최대한 폭넓게 준비되어야 한다. 딥 러닝 모형에 한국 아파트 데이터를 적용하여 학습시킨 이후에, 다양한 해석 방법론을 학습된 모형에 적용하여 데이터에 내재한 규칙을 발견할 수 있다. 이러한 과정을 통하여 딥 러닝을 활용한 건축 공간 연구를 수행할 수 있음을 확인하였다.

## 제 3 장

### 딥 러닝을 활용한 단위평면 분석 방법론

이 장에서는 한국 아파트 단위평면에 적용할 수 있는 딥 러닝을 활용한 분석 방법론을 수립한다. 이 연구는 딥 러닝을 활용하여 한국 아파트 전수조사 규모의 단위평면 분석이 가능한 정량적이고 재현가능한 분석 방법론을 개발하고자 한다.

건축 공간에 대한 정량적 분석 방법론의 대표로 공간분석이 있다. 공간분석은 연역적 방법론으로, 사회적 구조와 물리적 공간이 서로 영향을 미치는 관계에서 공간의 구조를 분석하기 위한 방법론을 논리적으로 도출한다. 반면 딥 러닝을 포함한 기계학습 방법론은 귀납적 방법론으로, 데이터에 존재하는 규칙을 찾아 낼 뿐, 어떠한 규칙과 결과를 도출되는가는 데이터의 성격과 구조에 달려있다.

이처럼 딥 러닝을 활용한 단위평면 분석 방법론은 분석 대상이 건축 공간의 배치라는 점에서는 기존 공간분석과 같다. 따라서, 사회적 구조와 중심성에 대한 관심을 반영한 데이터를 활용하여 유사한 결과를 도출하는 모형을 학습시킬 수 있다는 점에서 공간분석의 대안적 방법론으로 활용될 수 있는 가능성이 있다.

그러나 기계학습의 가장 큰 장점인 데이터에 따라 어떠한 분석에도 적용할 수 있다는 특징을 살려 공간분석의 전제에 얹매이지 않고 건축 공간에 대한 다양한 분석이 가능한 귀납적 분석 방법론을 구축하는 것이 이 연구의 목표이다. 이를 위하여, 이 장에서 개발하는 방법론은 건축 공간을 딥 러닝을 적용하여 분석하고, 그 결과에서 건축적 의의를 도출하는 과정에 집중한다.

### 3.1 건축 공간 모형화

이 절에서는 건축 공간에 대한 분석에 딥 러닝을 활용할 수 있도록 건축 공간을 모형화한다. 이를 위하여, 먼저 건축 공간 분석에 적합한 딥 러닝 모형을 선정 한다. 다양한 딥 러닝 모형의 특성을 파악하고, 딥 러닝 모형의 분석 대상과 공간분석의 공간 구조 모형을 비교하여 건축 공간을 분석하기에 적합한 딥 러닝 모형을 선정한다. 이후 선정된 딥 러닝 모형에 아파트의 주거공간 배치를 학습시킬 수 있도록 건축 공간을 모형화하는 방법을 수립한다.

딥 러닝은 1960년대 등장한 다층 퍼셉트론에서 이어지는 다양한 심층 신경망 모형을 포괄한다. 그 중에서 활발한 연구가 진행되고 있는 분야는 크게 합성곱 신경망 (convolutional neural network, CNN)과 순환 신경망 (recurrent neural network, RNN) 두 가지로 나뉜다.<sup>36</sup> 이 절에서는 두 모형 각각을 대상으로 분석 데이터의 요구조건을 파악하고, 그러한 요구조건에 가장 잘 맞는 공간 구조 모형을 대응시켜 단위평면 분석에 대한 활용 가능성을 검토한다.

합성곱 신경망은 격자 구조의 데이터 처리에 특화된 딥 러닝 모형이다. 시계열 데이터는 일정 시간 간격의 1차원 격자로, 이미지는 2차원 픽셀 격자로 나타내는 등, 격자 구조로 표현할 수 있는 다양한 분야에 대한 분석에 활용이 가능하다.<sup>37</sup>

---

<sup>36</sup>Goodfellow 등. (2016). Deep Learning. MIT Press.

<sup>37</sup>Goodfellow 등. (2016). Deep Learning.

공간분석에서도 건축 공간을 격자 구조로 재현 (representation) 하는 모형이 존재한다. 시각적 접근-노출 (visual accesses and exposure, VAE)은 공간의 시각적 구조를 분석하기 위하여, 공간 안의 모든 지점에서 다른 모든 지점을 볼 때 시야 안에 들어오는 면적의 합계를 해당 지점의 시각적 접근으로, 반대로 다른 지점의 시야 안에 들어가는 경우의 합계를 시각적 노출로 정의하였다. 이 때 '모든 지점'은 2차원 직교 격자점으로 조작적으로 정의되었다. 공간을 일정 간격의 격자로 표현하였기 때문에, 면적 또한 해당 영역 안에 포함된 지점의 수로 정의될 수 있었다. 물론 Archea의 본래 정의는 수학적 정의라기보다는 절차적 정의였으며, 이러한 논의는 직관적으로 격자 구조가 채용된 이후에 사후적으로 이루어진 것이다.<sup>38</sup>

이러한 격자 구조에 대한 직관적 접근은 이후에도 반복된다. 가시성 그래프 분석 (visibility graph analysis, VGA)은 한 지점에서 보이는 모든 영역을 나타내는 isovist를 활용하여 공간 안 모든 지점의 가시영역 (isovist)이 맺는 관계를 공간 구조로 모형화한다. 이 때, “공간 전체에 걸쳐 가시영역을 생성”하는 “가장 명백한 접근법은 어떤 일정한 간격을 두고 생성하는 것”이며, “이는 생성되는 위치가 어떠한 규칙적인 격자 (grid or regular lattice) 위의 점이 된다는 함의가 있다”.<sup>39</sup>

이렇게 건축 공간을 2차원 직교 격자로 재현하는 것은 공간분석에서 널리 활용되는 방법론이다. 따라서 합성곱 신경망은 2차원 평면 위에 표현된 건축 공간에 대한 분석에 적합하다. 특히, 2차원 이미지 인식을 위한 합성곱 신경망 모형은 많은 연구가 이루어진 주제로 다양한 모형이 존재한다. 따라서 건축 공간을 이미지로 표현하는 단계를 추가하는 것만으로 건축 공간 분석에 기존

<sup>38</sup>황용하, 최재필. (2003). 시각적 접근-노출 모델의 재고찰. 대한건축학회 논문집-계획계, 19(3), 11-18.

<sup>39</sup>Turner 등. (2001). From Isovists to Visibility Graphs: A Methodology for the Analysis of Architectural Space. Environment and Planning B: Planning and Design, 28(1), 103-121.

신경망 모형을 활용할 수 있는 장점이 있다.

또한, 합성곱 신경망은 이미 이미지 형식으로 표현된 건축 공간에 대한 분석에도 적합하다. 위성사진처럼 원래부터 이미지인 2차원 데이터도 있지만, 지도나 평면도 등 건축도시 공간에 대한 정보를 이미지 형식으로 표현한 데이터도 공개된 출처를 통하여 쉽게 구할 수 있다. 이미지에서 건축 공간에 대한 정보를 인식하는 것은 건축 공간 분석과는 또 다른 주제이지만, 앞으로 공개된 데이터를 활용한 연구가 늘어나면서 그 중요성이 높아질 것이다.

다른 하나의 딥 러닝 모형인 순환 신경망은 순차적인 데이터 분석에 특화된 딥 러닝 모형이다. 필기 인식이나 음성 인식 등 전체 데이터를 한번에 처리하는 것이 불가능하거나 길이를 미리 알 수 없는 데이터를 처리하는 데 적합하다. 순환 신경망에서 각 단계의 출력값은 이전 단계의 출력값의 함수로 표현된다. 따라서 순환 신경망은 시계열적으로 가까운 요소끼리 밀접한 관련이 있는 데이터를 분석하는 데 적합하다.

일반적으로 건축 공간의 재현에서는 시간적 차원이 고려되지 않으므로, 1차원적인 구조의 순환 신경망이 활용될 여지가 많지 않다. 그러나 드물게 건축 공간에 대한 시계열적인 모형화 사례도 존재한다. 이동 경로에 따른 가시영역의 변화를 분석하는 경우, 가시영역과 관련한 다양한 지표가 일정 간격으로 변화하는 것으로 모형화할 수 있다.

다만 가시영역 자체는 2차원 데이터이므로, 가시영역의 변화를 직접 분석하기 위해서는 2차원 데이터를 분석할 수 있는 합성곱 신경망을 순환적인 구조로 구축한 모형이 필요하다. 그러한 구조의 신경망 모형인 순환 합성곱 신경망 (recurrent convolutional neural network, RCNN)이 연구되고 있으나 일반 합성곱 신경망에 비해 발전 속도나 활용 면에서 두각을 보이지는 못하고 있다.

이 연구에서는 공개된 평면도를 활용하여 한국 아파트 전수조사가 가능한

분석 방법론을 구축하는 것을 목표로 한다. 이러한 분석의 대상은 이미지 형식이거나 이미지 형식으로 변환할 수 있는 2차원 평면 공간이다. 따라서 단위평면을 분석하기 위하여 합성곱 신경망을 활용할 수 있도록 건축 공간을 2차원 직교 격자 구조의 이미지로 표현한다.

합성곱 신경망에 입력되는 이미지는 공간적으로는 2차원 격자 구조이지만 한 지점의 색을 표현하기 위하여 깊이 차원을 함께 갖는다. 예를 들어, RGB 컬러 이미지는 빛의 삼원색인 빨강, 초록, 파랑 세 가지 색 각각의 이미지 평면을 가진다. 이러한 구조는 건축 공간에서 한 지점이 지니는 다양한 성격을 표현하는데에도 활용할 수 있다.

유사한 예로, 바둑 인공지능 알파고는 바둑 규칙에 따른 상황을 수십 가지 평면으로 표현한다. 각 지점에 놓인 돌의 색, 층 여부 등과 함께, 인접한 8지점의 수순, 활로의 수, 따내거나 따일 수 있는 돌의 수 등을 모두 각각의 평면으로 표현한다. 이렇게 하여 바둑판의 크기는  $19 \times 19$ 개 점에 불과하지만, 각 지점마다 11가지 특징 (feature)을 48차원에 걸쳐 표현하게 된다.<sup>40</sup> 마찬가지로 이미지 깊이 차원을 활용하여 건축 공간의 다양한 특성도 딥 러닝 모형에 입력하여 분석할 수 있다.

건축 공간을 모형화하는 과정에서 실제로 어떠한 특징이 포함되는지는 분석 설계와 가용한 데이터의 성격에 따라 달라질 수 있다. 예를 들어, VAE나 VGA 분석 결과는 그 자체로 딥 러닝 분석의 대상이 될 수 있다. 이러한 경우, 각 분석에서 도출된 다양한 지표값이 입력 이미지의 깊이 차원이 된다. 만약 BIM 데이터를 분석에 활용하는 경우, 각 지점별로 실 종류나 면적 등의 벡터 데이터를 래스터화 (rasterization) 하여 표현할 수 있다.

한편, 이미지 형태로 수집된 도면이나 위성사진 등을 활용하는 경우, 미리 전처

<sup>40</sup>Silver 등. (2016). Mastering the game of Go with deep neural networks and tree search. Nature, 529(7587), 484–489.

리를 적용하느냐에 따라 두 가지 방향이 가능하다. 하나는 전처리를 적용하지 않고 이미지를 그대로 입력하여 딥 러닝 모형에 종단간 학습 (end-to-end learning)을 적용하는 것이다. 이 경우 입력 데이터는 원본 이미지 자체가 되며, 사람이 눈으로 도면을 보고 공간을 파악하듯이 딥 러닝 모형이 색으로 표현된 이미지에서 건축 공간을 인식하도록 학습시킨다. 다시 바둑 인공지능에 비유하면, 바둑판에 놓인 돌의 배치와 수순만 주고 현재 형세를 알아서 학습하도록 시키는 것과 같다. 다른 하나는 사람이 먼저 이미지를 처리하고 건축 공간의 구조를 나타낼 수 있는 데이터를 딥 러닝 모형에 입력하는 것이다. 이 경우 건축 공간 모형에 포함되는 특징은 이미지에서 어떠한 정보를 어떻게 구성하느냐에 따라 다양하게 나타날 수 있다.

딥 러닝은 데이터에서 주어진 작업을 수행하는 방법을 배울 수 있을 뿐만 아니라, 이를 위해 필요한 특징을 추출하는 표현 학습 (representation learning)에도 뛰어나다. 따라서 이론적으로는 사람과 마찬가지로 종단간 학습을 통하여 이미지에서 주거공간의 배치를 바로 학습할 수 있다. 그러나 현실적으로는 이러한 방법을 적용하기 어려운 몇 가지 단점이 있다.

그 한 가지는, 종단간 학습에는 매우 많은 계산량이 필요하다는 것이다. 이미지 인식은 그 자체만으로도 충분히 어려운 문제이기 때문에 한번에 처리하는 것보다 여러 단계로 나누어 각자 다른 방법론을 적용하는 것이 훨씬 쉬울 수 있다. 다른 한 가지는, 딥 러닝 모형이 무엇을 학습할지 통제할 수 없다는 것이다. 딥 러닝 모형이 이미지를 인식하는 방식은 사람과 크게 다르기 때문에 연구자가 의도하지 않은 의외의 특징을 찾아 학습할 수 있다. 이러한 경우 연구자가 직접 입력 데이터의 전처리를 수행하는 것은 그 안에 담긴 정보를 선별하여 정확한 학습을 유도하는 효과가 있다.

따라서 이 연구에서는, 합성곱 신경망에 적합한 2차원 이미지의 형태로 건축 공간의 배치를 모형화한다. 특히, 전처리 과정에서 미리 주거공간과 관련된 정보만 선별하여 딥 러닝 모형이 주거공간의 배치를 정확하게 학습할 수 있도록

한다.

## 3.2 표현학습을 통한 주거공간 배치 분석

한국 아파트의 단위평면에서 나타나는 주거공간 배치 특성을 데이터에서 귀납적으로 학습하는 표현학습 (representation learning)을 통하여 분석한다. 딥 러닝 모형은 표현학습을 통하여 데이터에 잠재된 구조를 모형 내부에 재현한다 (represent). 딥 러닝이 큰 성공을 거둘 수 있었던 것은 이 방법론을 적용하여 다양한 작업에 필요한 정보를 데이터에서 잘 학습할 수 있었기 때문이다. 따라서 딥 러닝 모형이 한국 아파트 단위평면을 대상으로 적절한 작업을 수행하도록 학습시키면 그 학습 과정에서 단위평면의 주거공간 배치에서 분석하고자 하는 특성을 딥 러닝 모형이 도출하여 재현하도록 할 수 있다.

주거공간 배치 분석에 딥 러닝을 활용하는 이 연구에서는 딥 러닝 모형이 수행하는 작업 자체가 목적이 아니기 때문에, 어떠한 작업을 통하여 학습할 것인지를 정해져 있지는 않다. 따라서 딥 러닝이 수행할 수 있는 다양한 작업 중에서 가장 적합한 작업을 선정할 수 있다.

기계학습 및 딥 러닝 분야에서 가장 많이 연구되는 작업은 분류 (classification)이다. 다양한 패턴 인식 (pattern recognition) 문제는 분류 작업으로 표현될 수 있기 때문이다. 손글씨를 인식하여 숫자를 구분하는 MNIST나, 사진에서 사물을 식별하는 CIFAR-10, CIFAR-100 등, 대다수의 데이터셋이 분류 작업의 기계학습을 위하여 구축되었다. 딥 러닝의 발전 속도만큼 분류 문제의 규모도 성장해왔다. MNIST 데이터셋은 10가지 숫자를 쓴 손글씨 이미지로 이루어져 있지만, ImageNet10K 데이터셋은 약 1만 가지 사물의 사진을 담고 있다.

분류 작업을 수행하기 위하여 딥 러닝 모형이 학습하는 것은 데이터셋 안에 포함된 서로 다른 대상의 특징과 차이점이다. 각 대상만의 특징을 잘 학습하였

다는 것은 분류 작업을 정확하게 수행할 수 있다는 것을 보임으로서 증명된다. 딥 러닝은 분류 작업으로 표현된 이미지 인식 문제에서 다른 방법론과 비교할 수 없는 수준의 정확도를 보이고 있으며, 때로는 사람보다 정확한 경우도 있다.

이러한 딥 러닝의 학습 능력은 한국 아파트 단위평면에 대한 연구에서도 활용될 수 있다. 아파트에 대한 선행 연구에서도 분류 작업으로 표현될 수 있는 연구 주제가 많이 발견된다. 시기별 단위평면 계획의 변화는 많은 선행연구에서 활용되는 분석 대상이다. } 특정 사건이 건축 계획에 미친 영향을 파악하기 위하여 해당 사건 전후 시기 단위평면을 비교하는 경우도 있지만,<sup>41</sup> 많은 경우 각 시기별 단위평면에 대한 분석을 통하여 통시적인 변화 양상을 도출한다.<sup>42</sup> 반드시 시기에 따라 구분하지 않더라도, 다양한 기준에 따라 단위평면을 유형화하는 연구도 많다. 그 기준이 단위평면에 내재된 것이 아니라 위치한 국가,<sup>43</sup> 공급 방식,<sup>44</sup> 주동 형태<sup>45</sup> 등의 외부적 조건이라면, 서로 다른 집단 사이의 차이를 비교하여 외부적 조건이 단위평면에 미친 영향을 분석하는 방법론은

<sup>41</sup> 박인석 등. (2014). 전용면적 산정기준 변화와 발코니 용도변환 허용이 아파트 단위주거 평면설계에 미친 영향: 전용면적 60m<sup>2</sup> 와 85m<sup>2</sup> 평면의 실별 규모 변화를 중심으로. 한국주거학회논문집, 25(2), 27-36.

<sup>42</sup> Byun, Choi. (2016). A Typology of Korean Housing Units: In Search of Spatial Configuration. Journal of Asian Architecture and Building Engineering, 15(1), 41-48.; 이상진, 박소현. (2019). 부산 아파트 단지의 주동 평면형태의 변화 특성에 관한 연구. 대한건축학회 논문집-계획계, 35(8), 3-14.; 최병숙, 박정아. (2009). 여성관련 공간을 중심으로 본 서울지역 아파트의 공간구조 변화. 한국주거학회논문집, 20(4), 39-47.; 최재필. (1996). 공간구문론을 사용한 국내 아파트 단위주호 평면의 시계열적 분석-수도권 4LDK 아파트 단위주호 평면계획을 중심으로. 대한건축학회 논문집, 12(7), 15-27.; 최재필. (1996). 한국 현대 사회의 주생활양식의 변화-수도권 3LDK 아파트 주호 평면 계획의 변천을 중심으로. 대한건축학회 논문집, 12(9), 3-12.

<sup>43</sup> 김성규, 김경배. (2016). 한국과 중국의 아파트 평면계획 특성과 주생활 문화의 관계 연구. 한국도시설계학회지 도시설계, 17(5), 49-61.; 단경위, 신경주. (2012). 한국과 중국 아파트의 평면구성 비교. 한국실내디자인학회논문집, 21(5), 46-56.

<sup>44</sup> 최재필 등. (2016). 리모델링을 대비한 1 기 신도시 노후 공동주택의 대표 유형에 관한 연구. 대한건축학회 논문집-계획계, 32(4), 33-40.

<sup>45</sup> 배연희, 하미경. (2019). 최근 공동주택의 주동형태 및 단위세대 평면 유형에 관한 연구-2019년 살기 좋은 아파트를 중심으로. 한국실내디자인학회 논문집, 28(6), 86-95.

동일하게 적용될 수 있다.

그러나 딥 러닝 모형이 대상에 대한 이해를 갖춘 것과 그것을 사람이 이해할 수 있는 방식으로 설명하는 것은 별개의 문제이다. 마치 어떤 학생이 다양한 건축 양식을 꼭 집어 설명하지는 못하더라도 보면 바로 알아맞추는 경우와 같다. 딥 러닝은 모형의 높은 복잡도 때문에 특히 여러 기계학습 방법론 중에서도 가장 모형의 내부 표현 (internal representation)을 이해하기 어려운 편에 속한다. 이 때문에 딥 러닝 모형이 학습한 내용을 사람이 이해할 수 있는 형태로 표현하기 위한 연구도 활발하게 진행되고 있다.

앞에서 논의한 내용을 고려하여, 시기 등 다양한 기준에 따른 단위평면 계획의 차이를 분석할 수 있는 딥 러닝 모형을 선정한다. 딥 러닝 모형의 선정 기준은 크게 두 가지이다. 첫 번째는 이미지 형태로 정리된 단위평면 데이터셋에서 서로 다른 집단 사이에서 나타나는 주거공간 배치 계획의 차이를 학습할 수 있는 능력이 있어야 하며, 두 번째는 모형 내부에 학습된 단위평면 계획 특성을 사람이 해석할 수 있는 형태로 도출할 수 있어야 한다.

이를 위하여, 이 연구에서는 Zhou 등이 개발한 CAM (class activation mapping) 방법론<sup>46</sup>을 활용한다. CAM 방법론은 이미지 인식을 통한 분류를 수행하는 딥 러닝 모형에 대하여 각 이미지가 해당 범주로 분류된 이유를 시각화하는 방법론이다. 구체적으로, 이미지 안의 각 영역에 대하여 해당 범주의 특성으로 인식되는 강도를 측정하고, 이를 전체 이미지에 대하여 시각화한다. 이러한 과정을 통하여 딥 러닝 모형이 이미지 안의 어느 영역에서 해당 범주의 특성이 나타난다고 판단하는지를 알 수 있다.

그림 3.1은 이미지 분류 모형에서 각 이미지가 분류되도록 하는 특성이 나타나는 영역을 CAM으로 시각화한 것이다.<sup>47</sup> 윗줄의 이미지는 각각 양치질과 토질을 나타내는 사진으로, 딥 러닝 모형이 해당 행동으로 정확하게 분류하고 있다.

<sup>46</sup>Zhou 등. (2015). Learning Deep Features for Discriminative Localization.

<sup>47</sup>Zhou 등. (2015). Learning Deep Features for Discriminative Localization.

아랫줄에서는 각 이미지에 대한 CAM 분석 결과를 보여준다. 사진 안에서 어떤 영역이 딥 러닝 모형의 판단에 각 영역이 미친 영향의 크기를 색으로 표현하여, 가장 큰 영향을 준 영역을 빨간 색으로 나타낸다. 이러한 시각화를 통하여 딥 러닝 모형은 칫솔을 보고 양치질을 인식하고, 전기톱과 모자를 보고 톱질을 인식하고 있다는 것을 사람도 쉽게 이해할 수 있다. 이렇게 CAM 방법론을 통하여 사람이 쉽게 파악하기 어려운 딥 러닝 모형의 내부적인 구조를 파악할 수 있다.



그림 3.1: CAM을 통한 분류 근거 영역 시각화<sup>48</sup>

단위평면을 특정 기준에 따라 분류하는 딥 러닝 모형에 CAM 방법론을 적용하면, 각 범주별 단위평면의 주거공간 배치에서 나타나는 계획 특성을 시각화할 수 있다. 예를 들어, 특정 시기, 지역, 주동 형태의 단위평면을 특징짓는 계획 특성에 해당하는 영역을 강조하여 나타낼 수 있다는 의미이다. 연구자는 해당 영역에 표현된 주거공간 배치를 보고, 건축적 조건을 고려한 외재적 해석을 제공할 수 있다.

딥 러닝에 CAM을 적용하기 위하여 저자들은 딥 러닝 모형의 마지막 계층

<sup>48</sup>Zhou 등. (2015). Learning Deep Features for Discriminative Localization.

에 GAP (global average pooling) 계층을 적용하였다. 논문에서 저자들은 AlexNet, VGGNet, GoogLeNet 3가지 이미지 분류 모형에 대하여 GAP 계층을 적용하여 그 성능을 비교하였는데, 이 중 VGGNet과 GoogLeNet이 여러 기준 모두에서 유사하게 뛰어난 성능을 보였다.<sup>49</sup> 두 모형 모두 이 연구에 활용하기에 적합하지만, VGGNet의 신경망 구조가 더 단순하고 직관적인 장점이 있어<sup>50</sup> 이 연구에서는 VGGNet에 GAP 계층을 적용한 VGG-GAP 모형을 단위평면 주거공간 배치 분석 모형으로 선정한다.

주거공간 배치 분석 모형은 단위평면 데이터셋에서 단위평면을 기준에 따라 분류하는 과정을 통하여 각 범주별 주거공간 배치 특성을 학습한다. 따라서 이 연구의 방법론을 검증하기 위하여 구축되는 4장의 한국 아파트 단위평면 데이터셋은 단위평면의 주거공간 배치 데이터 뿐만 아니라, 분류를 통한 학습이 가능하도록 주거공간 배치 분류 기준이 될 수 있는 데이터를 함께 포함하여야 한다. 이를 데이터는 우선 이 연구에서 선정한 VGG-GAP 모형에 적용할 수 있도록 구축되어야 하지만, 다른 한편으로는 앞으로 개발될 다양한 딥 러닝 모형에 활용할 수 있도록 최대한 범용적인 형태로 구축한다.

### 3.3 딥 러닝을 적용한 단위평면 유형화

딥 러닝 모형이 기계학습을 통하여 도출한 잠재표현 (latent representation)에서 나타나는 다양한 계획 특성을 기준으로 단위평면을 유형화하는 방법론을 수립한다. 단위평면 분류 작업을 학습한 딥 러닝 모형에 잠재된 주거공간 배치 계획 특성을 추출하고, 각 계획 특성과 이에 해당하는 단위평면을 유형화한다. 이를 통하여, 이 연구에서 수립한 표현학습 기반 주거공간 배치 분석 방법론을 검증하고, 딥 러닝 기반 건축 공간 분석의 새로운 가능성을 탐색한다.

---

<sup>49</sup>Zhou 등. (2015). Learning Deep Features for Discriminative Localization.

<sup>50</sup>Simonyan, Zisserman. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition.

딥 러닝 모형 내부에서 입력값의 여러 특징을 인식한 결과인 잠재표현은 출력 층 직전 계층에서 수천 개의 노드가 활성화되는 값으로 표현된다. VGGNet에서는 4096개, VGG-GAP에서는 1024개의 노드가 분류 작업을 위하여 학습한 이미지 속 다양한 패턴을 인식한다. 각 노드는 서로 독립적으로 학습한 패턴과 일치하는 영역이 나타날 때 활성화된다. 즉, 딥 러닝 모형이 데이터셋에서 학습한 잠재표현은 수천 개의 독립적인 판단으로 구성된다.

CAM 방법론은 이러한 잠재표현을 활성화하는 영역을 최종 분류의 각 범주에 따라 중첩하여 분석하는 방법론이다.<sup>51</sup> 그림 3.2은 잠재표현의 각 노드를 활성화하는 영역을 중첩하여 해당 범주를 활성화하는 영역을 하나로 합성하는 과정을 보여준다.

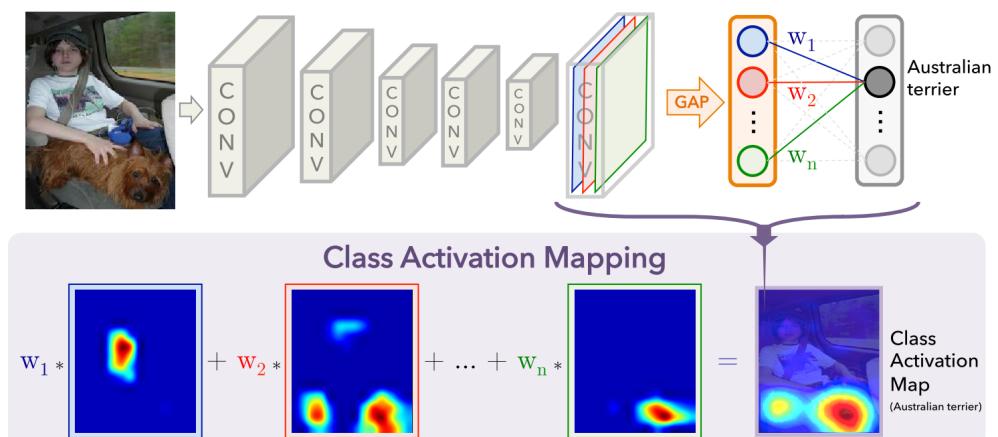


그림 3.2: CAM 시각화의 구조<sup>52</sup>

GAP 계층의 첫 번째 노드가 활성화될 때, 사진 속 대상이 오스트레일리안 테리어 견종이라는 판단을 강화하는 정도가  $w_1$ 로 표현되고, 두 번째 노드는  $w_2$ 로, 마지막 노드는  $w_n$ 으로 표현된다. CAM은 테리어 견종이라는 판단의 근거가 되는 영역을 시각화하기 위하여, 첫 번째 노드를 활성화하는 영역에

<sup>51</sup>Zhou 등. (2015). Learning Deep Features for Discriminative Localization.

<sup>52</sup>Zhou 등. (2015). Learning Deep Features for Discriminative Localization.

$w_1$ 의 가중치를 주고, 해당 영역이 테리어라는 판단을 강화하는 정도만큼만 강조되도록 한다. 사진 속에서 첫 번째 노드를 활성화하는 영역에 위치한 것은 테리어가 아니라 사람이다. 사진 속 사람의 등장은 테리어 견종을 인식하는데 도움을 주지 않으므로, 딥 러닝 모형은 학습 과정에서  $w_1$ 을 작게 조정하여 테리어와 관련 없는 패턴이 분류 판단에 영향을 미치지 않도록 한다. 따라서 첫 번째 노드의 활성화 영역은 최종 CAM 시각화에 크게 영향을 미치지 못한다. 반대로, 두 번째 노드는 테리어가 위치한 영역에서 나타나는 패턴을 잘 인식하고 있으므로,  $w_2$ 는 높은 값을 갖고, 두 번째 노드를 활성화하는 영역은 최종 CAM 시각화에 큰 영향을 미친다.

이런 식으로, 각각의 노드를 활성화하는 영역이 최종 분류 판단을 강화하는 정도만큼 강조된 시각화를 도출한다. 노드별 시각화 전체를 합산하면 딥 러닝 모형 전반에 걸쳐 테리어로 분류된 근거가 되는 영역을 나타내는 시각화가 완성된다.

3.2절에서 수립된 주거공간 배치 분석 방법론은 이러한 CAM 방법론을 적용하여 단위평면을 주어진 기준에 따라 분류하고 그렇게 분류된 이유에 해당하는 영역을 시각화할 수 있다. 예를 들면, 시기별 혹은 지역별로 한국 아파트 단위평면에서 나타나는 계획적 특성을 시각화하여 분석할 수 있다. 그림 3.3은 VGG-GAP 모형에서 특정 시기나 지역 분류에 영향을 미친 영역을 강조하는 CAM 시각화를 나타내는 과정을 보여준다.

그러나 단위평면 계획의 변화는 사진 속 사물을 인식하는 경우에서처럼 이미지 속 한 지점에 집중되어 나타나지 않을 것이다. 그보다는, 다양한 변화가 단위평면 전반에 걸쳐 여러 영역에서 나타날 것이다.

이러한 문제는 CAM 방법론의 목표가 딥 러닝 모형의 사물 인식 결과를 검증하는 것이라는 점에 내재되어 있다. 이미지 인식 딥 러닝 모형에 대한 CAM 시각화 결과에서는, 특정 사물이 위치한 영역이 강조되기만 하면 해당 사물의

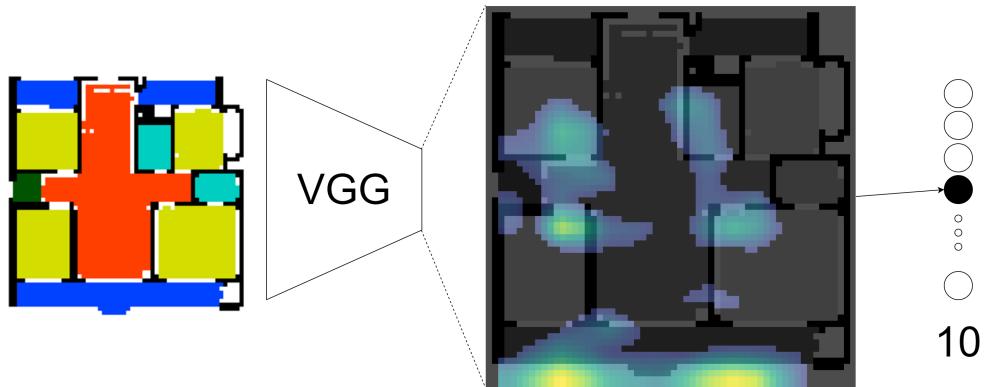


그림 3.3: 단위평면 CAM 시각화 구조

특징을 정확하게 학습하였다는 것을 검증할 수 있다. 그림 3.1에서 칫솔과 전기톱과 모자의 위치를 맞춘 것만 보아도 딥 러닝 모형이 이들 사물의 특징을 잘 인식한다는 것을 알 수 있는 것과 마찬가지다.

그러나 단위평면의 계획 특성은 칫솔과 전기톱처럼 명백하게 구분되지 않는다. 또한, 단위평면에서 나타나는 각 계획 특성은 사진 속 사물처럼 명백한 경계를 보이지 않는다. 따라서 단위평면의 특정 영역이 어떠한 계획 특성을 나타내는지를 설명하는 것은 사진 속 사물을 설명하는 것보다 훨씬 어려운 일이다. 그렇기 때문에, CAM 방법론을 적용하여 분류 기준에 따라 근거 영역을 도출하여도 해당 범주의 특징을 정확하게 해석하기 어렵다.

또한, 이 연구의 방법론은 분류 작업 자체를 수행하는 모형을 검증하는 CAM 방법론보다 한 단계 더 내려가, 그 과정을 통하여 주거공간 배치의 특성을 분석하는 것이다. 마찬가지로 이 연구의 방법론을 검증하기 위해서는, CAM이 분류의 근거가 되는 영역을 도출하는 것에서 한 단계 더 내려가 특정 범주의 계획 경향을 구성하는 계획 특성을 분리하여 분석할 필요가 있다.

주거공간 배치 계획 유형은 유사한 주거공간 배치가 나타나는 여러 단위평면과 단위평면의 해당 영역에서 활성화되는 잠재표현 노드의 연관관계에서 나타난

다. 각 배치유형의 특징은 해당 유형의 단위평면에서 해당 잠재표현 노드가 활성화되는 영역을 강조하는 시각화를 통하여 표현된다. 딥 러닝 모형이 수행하도록 학습한 분류 작업에서, 분류 기준에 따른 각 범주는 여러 계획 유형과 다양한 관계를 맺는다. 또한, 한 계획 유형이 여러 범주에 걸쳐 나타날 수도 있다.

이렇게 단위평면 객체와 잠재표현 속성 양쪽에 대한 군집화를 수행할 수 있는 방법론이 바이클러스터링 (biclustering, 양방향 군집화) 분석이다.<sup>53</sup> 모든 객체의 속성 값을 행렬로 표현한 데이터 행렬에서, 서로 연관된 일부 행과 일부 열이 만드는 군집을 바이클러스터 (bicluster)라고 한다. 바이클러스터링 분석은 전체 데이터 행렬을 바이클러스터로 분할하여, 높은 연관을 보이는 객체와 속성을 짹짓는다 (그림 3.4).

바이클러스터링 분석은 어떠한 바이클러스터를 찾는지에 따라 다양한 알고리즘이 존재하나, 일반적으로 평균 수준 대비 높거나 낮은 값의 바이클러스터를 찾게 된다. 이 연구에서는 유사한 주거공간 배치가 나타나는 일부 단위평면을 대상으로 일부 잠재표현 노드가 높은 활성화를 보이는 계획 유형을 찾고자 하므로, 평균 수준 대비 높은 값을 보이는 바이클러스터를 찾는 경우에 해당한다.

이 연구에서는 이러한 바이클러스터를 찾는 알고리즘 중 하나인 spectral co-clustering을 적용한다.<sup>55</sup> 이를 통해, 딥 러닝 모형이 학습한 잠재표현에 대한 바이클러스터링 분석을 통하여 단위평면 유형화를 수행한다. Spectral co-clustering은 각 행에서 평균 대비 높은 값을 보이는 열과 각 열에서 높은 값을 보이는 행을 짹짓는다. 이러한 과정을 통하여 딥 러닝 모형이 학습한 잠재표현에서 특정한 단위평면에서 활성화되는 노드와 그 노드를 활성화하는

<sup>53</sup>Scikit-learn developers. (2020). Biclustering.

<sup>54</sup>Scikit-learn developers. (2020). A demo of the Spectral Co-Clustering algorithm.

<sup>55</sup>Dhillon. (2001). Co-clustering documents and words using bipartite spectral graph partitioning. In Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining (pp 269–274).

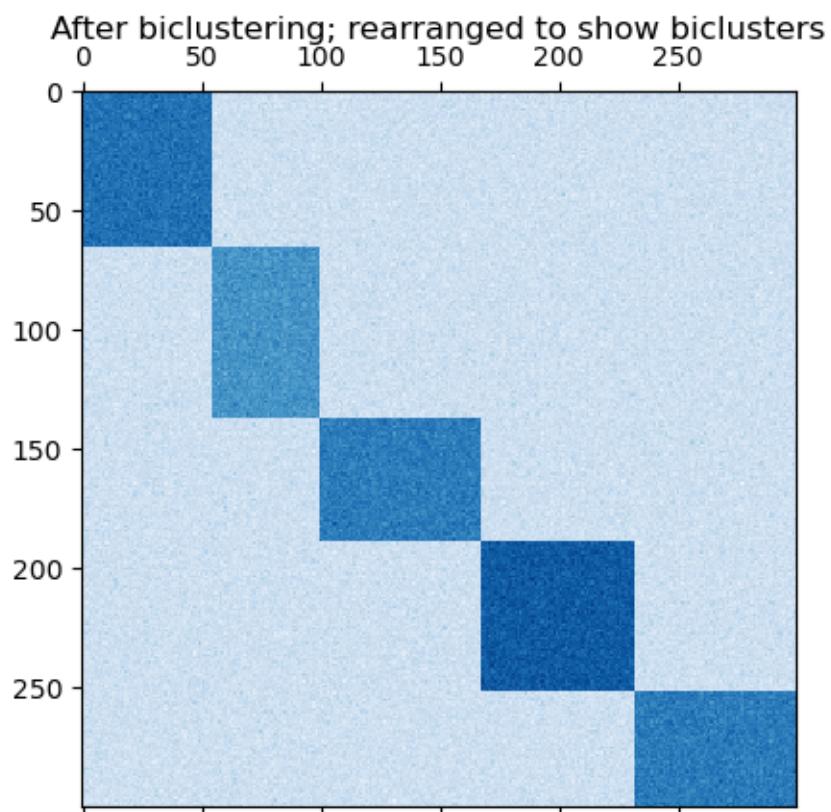


그림 3.4: 바이클러스터링 분석 예시<sup>54</sup>

단위평면을 양방향으로 군집화하고, 둘을 묶어 하나의 주거공간 배치유형으로 도출하게 된다.

이 때, 모든 행과 열은 단 하나의 바이클러스터에만 속한다 (그림 3.4). 단위평면 유형화에 이를 적용하면, 하나의 단위평면은 하나의 배치유형에만 속하고, 각 잠재표현 노드 또한 배치유형별 모형 중 하나에만 속하게 된다. 물론 현실에서는, 하나의 단위평면 안에서 여러 주거공간에 대한 다양한 배치 계획의 변화가 동시다발적으로 일어날 수 있다. 따라서 이 연구에서 이루어지는 단위평면 유형화는 해당 계획 유형이 가장 명확하게 나타나는 단위평면을 선별한다는 의미로 한정하여 해석하여야 한다.

바이클러스터링을 통하여 도출된 단위평면 계획 유형을 해석하기 위하여, 해당 유형의 단위평면에서 해당 유형 노드를 활성화되는 영역을 시각화한다. 이는 CAM에서 클래스 분류를 기준으로 시각화하는 대신 바이클러스터를 기준으로 잠재표현을 분리하여 시각화하는 것이다. 이 연구에서는 이러한 시각화 방법론을 BAM (bicluster activation map) 이라고 한다.

BAM은 CAM을 산출하는 수식에서  $w_k$ 를 실제 딥 러닝 모형의 클래스 분류 가중치 대신 바이클러스터 포함 여부로 바꾼다. 또한, CAM 분석에서 이미지의 실제 범주와 분류된 클래스는 다를 수 있기 때문에, 같은 이미지에 대하여 다양한 클래스에 대한 CAM 시각화가 도출될 수 있지만, BAM 분석에서는 단위평면과 동일한 바이클러스터 유형에 해당하는 시각화만 도출된다. 그러나 BAM 분석 결과의 해석은 CAM과 마찬가지로, 강조된 영역이 해당 유형의 계획 특성을 보여준다는 것을 나타낸다.

단위평면의 유형화는 바이클러스터링과 BAM 시각화를 순환적으로 진행하는 과정에서 구체화된다. 바이클러스터링에서는 유형 분류가 도출되지만, 각 유형이 어떠한 계획 특성을 보여주는지를 분석하기 위해서는 BAM 시각화가 필요하다. 한편, BAM 시각화를 통해 해석된 각 유형은 다시 바이클러스터링

의 군집화 결과를 활용하여 분석된다. 이러한 순환적 과정을 통하여 딥 러닝 모형이 학습한 단위평면의 주거공간 배치를 귀납적으로 도출된 주거공간 배치 계획 특성으로 유형화한다.

### 3.4 방법론 검증

이 장에서 수립한 딥 러닝을 활용한 주거공간 배치 분석 방법론을 한국 아파트 단위평면의 진화과정 분석에 적용하여 검증한다.

3.2절에서 분석한 바와 같이, 한국 아파트 단위평면에 대한 선행 연구는 많은 경우 분류 작업으로 환원할 수 있다. 단위평면 계획의 시간에 따른 변화를 시기 구분을 통하여 분석하는 경우 시기별 분류 작업을 수행하는 것과 같다. 국가나 지역에 따른 계획 특성 비교는 지역별 분류로, 주동 형식이나 분양 방식별 단위평면의 차이도 분류 작업으로 환원될 수 있다.

단위평면의 주거공간 배치에 대한 연구에서도 이러한 점은 마찬가지다. 차이점은 분류 작업의 기준이 단위평면에 내재하며, 연구자에 의해 추가적인 분석이 수행되어야 한다는 점이다. 전통적인 연구 방법론에서는 그 분류 기준을 연구자가 연역적으로 수립하고, 그 기준을 아파트 단위평면에 적용하여 분석한다. 단위평면의 분류는 그런 다음에야 연구자가 수립한 기준에 따라 이루어진다. 이와 같은 연구의 초점은 분류된 각 범주의 단위평면 사례의 시기나 지역에 따른 분포에 집중된다.

그러나 이 연구를 비롯하여, 기계학습 방법론을 적용한 연구에서는 단위평면에 내재한 특성이 기계학습 알고리즘에 의하여 귀납적으로 학습된다. 이 연구에서 수립한 방법론에서는 딥 러닝 모형이 연구자가 미리 분류한 데이터셋에서 분류 작업을 위한 기준을 학습하는 과정에서 단위평면의 주거공간 배치에 대한 분석이 이루어진다. 이 경우 분류 기준은 데이터셋 자체에서 도출되므로, 분류가 이루어진 이후 시기나 지역 등의 따른 분포를 분석하는 대신, 단위평면 외부의

요인에 따른 분류 작업을 먼저 수행한다. 그 이후에 분류 작업을 수행하면서 학습된 분류 기준, 즉 주거공간 배치 특성과 외부 요인에 따른 계획의 차이점을 분석한다.

한국 아파트에 대한 선행 연구 중에는 시기별 변화를 분석하는 연구가 가장 많다. 특히 단위평면 계획의 공간적 특성을 분석하는 연구는 시간의 흐름에 따른 공간 계획의 변화를 분석하거나, 탑상형 주동의 등장이나 발코니 확장 허용과 같은 특정 사건이 발생한 시점을 기준으로 전후 시기 평면 계획의 차이를 분석한다. 이를 통해 한국 아파트에서 나타나는 단위평면 계획의 진화 과정을 연구한다.

이렇게 다양한 선행 연구에서 연구된 한국 아파트 단위평면의 진화과정을, 이 연구의 방법론을 적용하여 분석하고 그 결과를 선행 연구와 비교하는 과정을 통하여, 이 연구에서 수립한 주거공간 배치 분석 방법론을 검증할 수 있다. 딥 러닝 모형에 한국 아파트의 단위평면 전수를 시기에 따라 분류하는 작업을 훈련시키고, 그 과정에서 딥 러닝 모형이 학습한 시기별 단위평면의 주거공간 배치 특성을 유형화하여, 한국 아파트 단위평면의 시간에 따른 진화과정을 귀납적으로 도출한다. 이 분석 결과를 선행 연구에서 도출된 다양한 발견과 비교하여 이 연구에서 수립한 방법론의 실효성을 검증한다.

4장에서는 한국 아파트에 대한 분석을 수행하는 데 필요한 한국 아파트 단위평면 전수에 대한 단위평면 데이터셋을 구축한다. 3.1절에서 논의한 바와 같이, 이 연구에서 선정된 VGG-GAP 모형을 비롯한 합성곱 신경망 모형에 단위평면의 주거공간 배치를 입력시켜 분석할 수 있도록, 수집된 단위평면에서 주거공간 배치를 인식하여 데이터셋으로 구축한다. 마찬가지로, 시기별 단위평면의 분류 작업을 딥 러닝 모형에 학습시키기 위하여 필요한 단위평면의 계획 시기를 범주화하여 포함한다. 또한, 지역 등 다른 조건에 따른 단위평면 계획 및 진화과정의 차이를 분석할 수 있도록 연관된 정보를 함께 수집하여 포함한다.

5장에서는 이 연구의 주거공간 배치 분석 방법론을 4장에서 구축된 데이터셋에 적용하여 한국 아파트 단위평면의 진화과정 분석을 수행한다. 3.2절에서 논의한 바와 같이, CAM/BAM 시각화를 가능하게 하는 합성곱 신경망 모형인 VGG-GAP 모형에 시기별 단위평면을 분류하는 작업을 훈련시킨다.

이 과정에서 딥 러닝 모형이 학습한, 단위평면 계획의 시기별 변화와 관련된 주거공간 배치 특성을 기준으로 한국 아파트 단위평면을 유형화한다. 3.3절에서 수립한 BAM 시각화를 적용하여, 각 유형의 주거공간 배치 특성을 해석한다. 이어서 각 유형별 단위평면 계획의 시기별, 지역별 분포에 대한 분석을 통하여 한국 아파트의 단위평면 계획의 진화과정을 분석한다.

이러한 과정을 통하여 도출된 진화과정을 선행 연구와 비교하여, 이 연구에서 수립한 방법론의 실효성을 검증한다. 시기 구분과 단위평면의 주거공간 배치만 주어진 상태에서 귀납적으로 분석된 결과를 선행 연구의 연역적 분석 결과와 비교하여, 귀납적 분석을 통해서도 한국 아파트의 진화과정이 발견될 수 있는지 확인한다. 이를 통하여, 딥 러닝을 활용한 주거공간 배치 분석 방법론의 가능성과 한계를 파악한다.

또한, 선행 연구와의 비교에서 공통점을 찾는 것에 그치지 않고, 이 연구에서 개발한 귀납적인 건축 공간 분석 방법론을 통하여 새롭게 발견된 내용에 대해서도 분석한다. 이 연구에서 딥 러닝을 적용한 분석은 단위평면 유형화에서 끝나고, 귀납적으로 정리된 정보에서 의미를 찾는 것은 다시 연구자의 몫으로 남는다. 이러한 분석 과정을 통하여, 귀납적 방법론을 적용한 건축 연구에서 연구자의 역할은 무엇인지 살피고, 향후 기계학습과 딥 러닝을 적용한 귀납적 건축 공간 분석이 나아가야 할 방향을 검토한다.

## 제 4 장

### 한국 아파트 단위평면 데이터셋

이 장에서는 딥 러닝을 적용한 단위평면 분석 방법론을 검증하기 위하여, 한국 아파트 단위평면 분석에 필요한 데이터셋을 구축한다. 데이터셋 (dataset, 정보집합물)은 “구조화된 데이터 개체들의 식별 가능한 집합”을 말한다.<sup>56</sup> 개별 자료가 체계적으로 정리되어 그 전체가 하나로서 인식되는 모든 자료의 집합은 데이터셋이 될 수 있으나, 현재는 그중에서도 특히 전자적으로 저장 및 공유되어 컴퓨터를 이용한 분석에 활용될 수 있는 것을 지칭한다. 마찬가지로, 한국 아파트 단위평면 데이터셋은 전국 아파트의 평면유형에 대한 개별 자료를 딥 러닝 분석에 활용할 수 있는 형태로 가공한 자료 집합을 의미한다.

2장에서 살펴본 바와 같이, 딥 러닝 모형의 학습에는 매우 많은 양의 데이터가 필요하며, 따라서 한국 아파트 전반에 대한 전수조사가 필수적이다. 마지막 단계에서 딥 러닝 분석의 결과를 몇몇 대표 평면으로 압축하여 제시하더라도, 그 대표 평면의 선정과 분석 결과는 한국 아파트 사례 전수에 대한 분석을 통해서만 도출될 수 있다. 따라서, 이 연구에서는 한국 아파트 전수조사를 통하여 전체 단위평면의 주거공간 배치를 딥 러닝 분석에 활용할 수 있는 형태로

---

<sup>56</sup>Project Open Data. (2014). Project Open Data Metadata Schema v1.1.

변환한 데이터셋을 구축한다.

한국 아파트 전수에 대한 데이터셋을 구축하기 위하여, 가장 먼저 공개적으로 접근할 수 있는 한국 아파트 단위평면과 관련된 데이터를 수집하고 정리한다. 그러나 이렇게 공개된 평면도는 품질이 보장되지 않으며, 도면의 다양한 요소가 갖춰져 있지 않거나 사례에 따라 빠지는 경우가 있다. 따라서 수집된 평면도에서 추출할 수 있는 주거공간 배치 관련 정보를 분석하고 이를 인식하기 위한 알고리즘을 개발하여 적용한다.

단위평면에서 인식된 주거공간 배치는 딥 러닝 분석에 적용할 수 있도록 정규화한다. 평면이 배치된 방향과 축척을 통일하고, 서로 다른 단위평면의 유사성과 차이를 분석할 수 있는 분석 영역을 설정한다. 단위평면과 연관된 기타 데이터에 대해서도 정규화 과정을 거쳐 평면도와 함께 딥 러닝 분석에 활용할 수 있도록 한다.

이렇게 수집되어 처리된 한국 아파트 단위평면의 주거공간 배치 관련 데이터를 딥 러닝에 적합한 형태의 데이터셋으로 구축한다. 이렇게 구축된 단위평면 데이터셋을 활용하여 한국 아파트 단위평면 공간에 대한 딥 러닝 분석이 가능하도록 한다.

이러한 과정은 모든 단위평면에 대하여 일관되게 적용하여야 한다. 따라서, 전체 구축 과정을 개별 평면에 대한 수작업 없이 자동으로 처리할 수 있도록 한다. 또한, 이러한 방식을 통하여 개별 평면에 대한 주관적 판단을 배제하고, 누구라도 동일한 작업을 수행하여 데이터셋 구축 과정을 재현할 수 있도록 한다.

## 4.1 단위평면 자료 수집

이 연구에서 수집하는 한국 아파트 단위평면 자료는 전국에 위치한 아파트의 단위평면 전수에 대한 평면도 및 기타 배치 계획 관련 정보이다. 딥 러닝 분석에 필요한 최대한의 사례를 수집하기 위하여, 전국 아파트에 대한 자료를 공개적으로 제공하는 출처를 수집 대상으로 한다.

네이버 부동산은 아파트를 비롯한 전국 부동산에 대한 정보를 공개적으로 제공하고 있으며, 그 안에는 단위세대 평면유형의 평면도와 단지 및 단위평면의 계획 관련 정보가 포함된다. 네이버 부동산에서 평면유형은 동일한 단위평면으로 계획된 단위세대를 말한다. 대칭 반전된 단위평면 또한 동일한 평면유형에 해당하며, 그 중 한 방향에 대한 평면도만 제공된다.

네이버 부동산에서 제공하는 평면도는 단위세대의 실내 계획을 일정한 크기의 이미지 안에 표현한다. 고정된 축척 없이 이미지 안에 단위평면이 가득 차도록 표현하기 때문에, 단위평면의 규모나 형태, 함께 표현되는 코어와 인접 단위세대 등에 따라 단위평면의 축척은 평면도에 따라 크게 달라진다. 마찬가지로, 단위평면의 방향 또한 다양하게 나타난다. 반전된 단위평면을 하나의 평면 유형으로 취급하지만, 일정한 규칙 없이 평면유형에 따라 서로 다른 방향의 단위평면이 표현된다.

평면에 표현되는 정보는 단위세대 실내 구조 및 주거공간 배치를 중심으로 각 실의 이름, 가구 배치, 면적, 치수 등의 정보가 부가된다. 일부 평면에서는 공용 공간과 인접 세대가 함께 표현되기도 한다. 각 실의 성격에 따라 색으로 표현되는데, 현관은 흰색과 회색의 체크 무늬, LDK는 갈색 마루 무늬, 침실은 황갈색, 발코니는 연미색, 화장실은 하늘색으로 나타낸다. 기타 설비 공간, 수납 공간 등을 회색으로 표현하는 경우가 있고, 가구나 출입문 같은 색을 채우지 않는다. 그림 4.1은 다양하게 표현된 평면도의 예시이다.

<sup>57</sup>NAVER Corp. (2017). 네이버 부동산.



그림 4.1: 평면도 예시<sup>57</sup>

네이버 부동산에서는 평면도 외에도 다양한 계획 관련 정보를 제공한다. 이러한 정보는 평면유형이 속한 단지의 준공연도, 지역, 세대 수, 층수 등 정보와, 평면유형의 전용면적, 침실 및 화장실 수, 주동 형식 등을 포함한다. 이러한 정보는 평면도와 함께 단위평면의 차이와 변화를 분석하는 데 활용될 수 있다.

네이버 부동산에서는 2017년 5월 24일 기준 전국 아파트 29,663 단지의 121,258 평면유형에 대한 정보를 제공하며, 그 중 51,497개 유형은 평면도를 함께 공개하고 있다.<sup>58</sup> 이 연구에서는 네이버 부동산에서 평면도를 제공하는 51,497 평면유형에 대하여, 평면도 및 계획 관련 정보를 수집하였다. 수집된 자료의 시도별, 준공연도별 분포는 그림 4.2와 같다.

## 4.2 주거공간 배치 인식

수집된 한국 아파트의 단위평면에서 나타나는 다양한 오류에 견고한(robust) 주거공간 배치 인식 알고리즘을 개발한다.

<sup>58</sup>NAVER Corp. (2017). 네이버 부동산.

<sup>59</sup>NAVER Corp. (2017). 네이버 부동산.

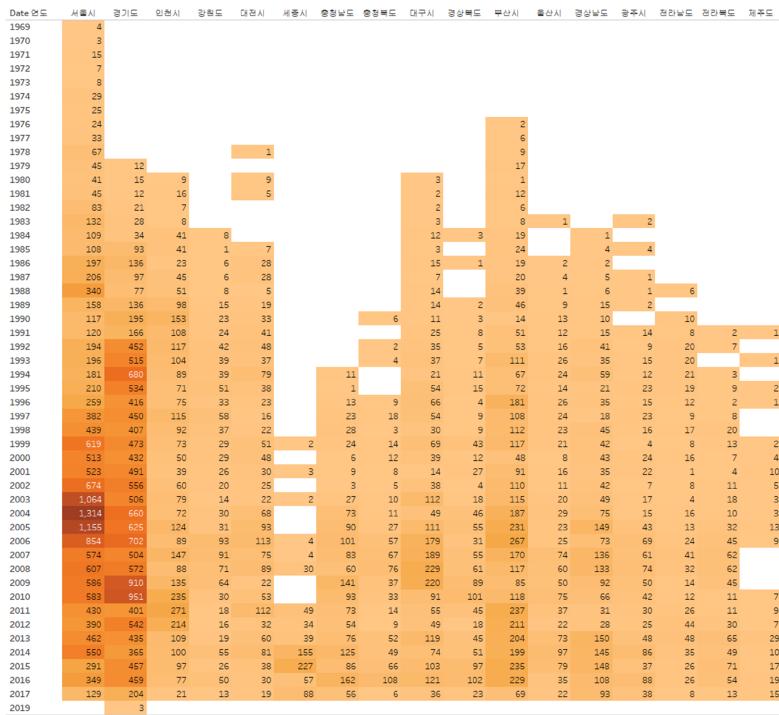


그림 4.2: 시도별 연도별 평면도 수록 건수<sup>59</sup>

#### 4.2.1 단위평면 분석 전략

한국 아파트의 단위세대는 대부분 단층이지만 복층 단위세대도 일부 존재한다. 그러나 단위평면에 복층 공간을 표현하는 방식은 일관되게 나타나지 않는다. 주거공간 배치 인식 알고리즘은 2차원 평면에 일관되게 표현되는 단층 단위세대를 대상으로 한다.

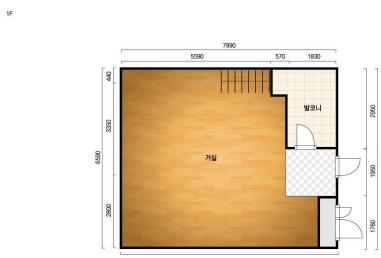
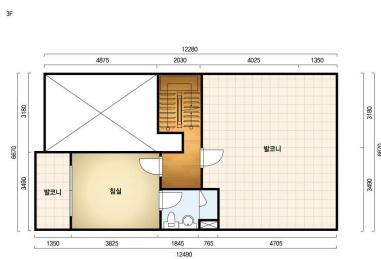
벽식 구조가 대부분인 한국 아파트에서 중요한 요소인 벽체는 단위평면에서 명확하게 나타나지만 정확하게 표현되지는 않는다. 대부분의 평면도에서 세대 경계를 단순히 모두 벽으로 표현하면서 개구부가 표현되지 않는다. 그러나 모든 단위평면에서 일관된 것은 아니고, 일부 평면은 개구부를 정확하게 나타내기도 한다. 문이나 막히지 않은 공간 경계가 벽으로 그려지는 오류도 종종 발생한다. 이를 주어진 정보대로 인식한다면 그 세대는 창이 없는 ‘먹방’으로 보일 것이다. 그렇기 때문에 벽체에 의존하여 공간 구조를 인식하는 것은 높은 빈도로 오류를 발생시키게 된다.

각 실에 대한 정보는 실 이름의 형태로 제공되기도 한다. 그러나 실 구분 기준이나 실 이름은 평면도마다 달라진다. 현관이나, 드레스룸, 발코니, 다용도실 등 규모가 작은 부속실의 경우 어떤 평면에서는 이름이 제공되고 다른 평면에서는 제공되지 않기도 한다. 일부 평면에서는 실별 면적까지도 제공하지만, 대다수의 평면에서는 그렇지 않아 단위평면 인식에 도움이 되지 않는다.

사람은 도면에 다양한 오류가 있어도 어려움 없이 실제 내용을 파악할 수 있다. 그러나 컴퓨터는 주어진 입력을 있는 그대로 인식하기 때문에 현실과 동떨어진 결과를 내놓는다. 이렇게 컴퓨터가 도면을 인식하려고 할 때는 치명적인 오류가 도면을 작성하고, 검수하고, 공개 후에는 오류를 신고받아 수정하는 여러 절차를 거친 이후에도 남아있다는 것은 그러한 오류에도 불구하고 많은 사람이 평면도를 읽는 데 큰 지장이 없었다는 뜻이다. 발코니 창호가 벽체로 그려져

---

<sup>60</sup>NAVER Corp. (2017). 네이버 부동산.



(가) 정렬되지 않은 복층 평면도



(나) 실 이름 오류 (오른쪽 침실 누락, 현관 및 주방/식당 오류)      (다) 벽체 및 출입문 오류 (오른쪽 위 침실)

그림 4.3: 단위평면 오류 예시<sup>60</sup>

있더라도, 사람은 발코니가 무엇인지, 아파트가 대체로 어떻게 생겼는지를 선형적으로 알고 있으므로 깨닫지도 못하는 사이에 주어지지 않은 정보를 이용해 도면을 고쳐 인식한다. 마찬가지로, 문 없이 벽으로 막힌 방이 단위평면에 존재해도 사람은 작성자의 의도를 이해하고, 심지어 문이 있을 위치를 추측할 수도 있다. 이는 사람이 평면도를 읽고 주거공간 배치를 인식하는 방식이 도면 정보를 있는 그대로 받아들이는 것과는 다르다는 의미이다.

이러한 오류가 모든 도면에 일관되게 유지된다면 알고리즘에 그러한 특성을 반영하여 교정할 수도 있다. 그러나 세대 경계의 벽체 표현처럼 의도적인 표현이 대다수 평면도에서 일치하는 경우에도 모든 평면이 통일되어 있지는 않으며 각 도면에는 의도적인 표현과 실수를 아울러 다양한 오류가 발생한다. 현실적으로 모든 도면이 오류 없이 그려져 있다고 가정할 수 없기 때문에, 평면도에서 주거공간 배치를 인식하려는 알고리즘도 이러한 한계를 반영한다.

반대로, 사람들이 가장 먼저 인식하고 고칠 필요를 느끼는 오류는 검증 과정에서 제일 쉽게 걸러질 것이므로, 모든 단위평면에서 가장 일관되게 오류 없이 표현되는 정보야말로 사람이 평면도를 인식하기 위해 가장 핵심적으로 의존하는 정보가 된다. 따라서 모든 평면도에서 주거공간 배치를 인식할 수 있는 알고리즘을 개발하기 위하여 사람이 주거공간 배치를 인식하고 재구성할 때 필요한 최소한의 정보만을 활용하여 주거공간 배치를 인식하고 데이터셋으로 구축한다.

이미지 파일 형식으로 제공되는 평면도를 분석하고 처리하기 위하여 파이썬과 영상처리 라이브러리인 OpenCV를 활용한다.<sup>61</sup>

---

<sup>61</sup>Bradski. (2000). The OpenCV Library. Dr. Dobb's Journal of Software Tools.

#### 4.2.2 단위세대 및 실 영역 인식

평면도에서 각 실을 구분짓는 가장 중요한 정보는 벽과 개구부이지만 모든 평면도에서 개구부가 정확하게 표현되어 있지 않다는 점이 걸림돌이다. (그림 4.3다) 도면 작성 과정에서 문을 빠뜨리면 벽으로 둘러싸인 어느 곳에도 연결되지 않은 방이 그려진다. 물론 현실에 존재하는 아파트에는 그런 일이 일어날 수 없고, 그 방은 어딘가로 연결되겠지만, 출입문이 빠진 평면도에서 추가적인 맥락 없이 정확한 위치를 재구성하는 것은 불가능하다. 따라서 개구부에 의존하지 않고 실내 영역을 인식할 필요가 있다.

수집된 단위평면에서 가장 명확하게 구분되고 오류 없이 제공되는 정보에 해당하는 것은 색으로 표현된 실 영역이다. 평면도는 단위세대 내부의 각 공간을 외부와 다른 색으로 채워 나타내고 있으며, 특히 현관, LDK (거실 및 주방, 식당), 침실, 발코니, 화장실은 각 실만의 색으로 일관되게 표현되고 있다. 따라서 주거공간 배치 인식의 첫 단계는 색으로 표현된 공간 정보를 이용해 단위세대 내부 영역을 인식하는 것이다.

그러나 실내 영역의 색도 해당 영역을 정확하게 반영하고 있지는 않다. 각 실의 영역이 모두 실내 영역의 색으로 채워진 것이 아니기 때문이다. 가구, 출입문, 부속 실 등은 단위세대 실내에 속하지만 실내 영역의 색으로 표현되지 않는다. 따라서 색으로 제공되는 정보를 참고하여 실내 실 영역의 정확한 경계를 파악하고 전체 단위세대 영역을 인식하는 과정이 필요하다.

가장 먼저, 실외가 확실한 영역을 분리하여 제외한다. 흰 배경과, 단위세대와 분리되어 있는 치수 등을 제외하고, 실내 영역의 색에 연결되어 있는 영역을 단위세대 영역의 후보로 삼는다.

단위세대 실내와 연결되어 있더라도, 계단실 코어가 평면도에 같이 표현되어 있는 경우 그 부분은 실외로 분류한다. 벽과 현관, 발코니 등으로 공간을 나누었을 때 실내 영역의 색과 연결되지 않고 분리되는 큰 공간이 있다면 그 공간은

공용 공간으로 판단하고 실외가 명백한 영역에 포함한다.



(가) 평면도 원본<sup>62</sup>



(나) 실외가 확실한 평면 영역

그림 4.4: 단위평면 실외 영역 인식

다음으로, 각 실의 경계 중에서 벽체를 제외한 실 경계 후보를 식별한다. 면적을 가지는 벽체는 쉽게 인식이 가능하며, 확실하게 각 실의 경계를 이룬다. 그러나 공간적으로 이어진 실 간의 경계를 인식하는 것은 단순하지 않은 문제이다. 각 실은 그 성격에 따라 색으로 구분되어 표현되지만, 그 실의 모든 공간이 같은 색으로 표현된 것은 아니다. 출입문이나 가구 등 다른 색으로 표현된 부분도 해당 실의 영역이므로, 그 영역이 인접한 실 중 어느 쪽에 속하는지를 판단할 필요가 있다. 따라서 벽체가 아닌 실 경계를 인식하는 과정이 필요하고, 이 단계에서는 그러한 경계가 될 수 있는 후보를 식별한다. 먼저, 실 경계를 표현하는 검은색 선을 인식한다. 평면도 이미지는 크기를 줄이기 위한 압축 과정에서 가느다란 선과 같은 형태는 왜곡될 수 있으므로, 선의 형태를 인식하는 것은 어렵다. 따라서 벽체를 제외한 검은색 영역을 검은색 선으로 인식한다. 이렇게 인식된 검은색 선 중에는 실제 선이 아닌 실 이름 및 치수 표기나, 검은색 실선이지만 실 경계가 아닌 출입문 및 가구 표기 등이 있다. 이 중에서 실제 실 경계가 될 수 있는 후보를 식별하기 위하여, 벽체로 구분된 실내 공간의 연결관계를 분석한다. 평면도의 형태학적 골격 (morphological skeleton)은 벽체를 제외한 공간의 형태에서 중심선만 남긴 것이다. (그림 4.5나) 형태학적 골격은 공간을

<sup>62</sup>NAVER Corp. (2017). 네이버 부동산.

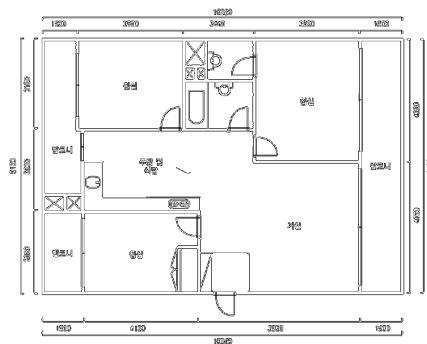
분절하는 실 경계를 식별하기에 적합한 도구이다. 실 경계를 식별하기 위한 전제는 실 경계는 두 실 사이를 연결하는 형태학적 골격과 교차하면서 동시에 벽체에 인접한다는 것이다. 이러한 조건을 만족하는 검은색 선만 선정하여 실 경계의 후보로 인식한다. (그림 4.5) 그림 4.6나는 실 경계 후보로 구분된 개별 영역을 나타낸다. 출입문이나 가구 등 실 경계 후보 중에도 실제로 실 경계가 아닌 경우가 있기 때문에, 여러 영역이 하나의 실에 속할 수 있다.

다음 단계에서는 각 실을 나타내는 색을 기준으로 실내가 확실한 영역을 찾고, 실내 공간을 파악하기 위한 핵으로 삼는다. LDK (거실 및 주방, 식당), 침실, 발코니, 화장실은 각 실의 색이 다른 실의 색과 색상, 채도, 명도로 구분되므로, 해당 영역을 쉽게 분리할 수 있다. 현관은 배경과 부속 실 등에서 나타나는 것과 같은 흰색과 회색으로 표현되므로, 두 색의 체크 무늬를 인식하여 현관 영역을 분리한다.

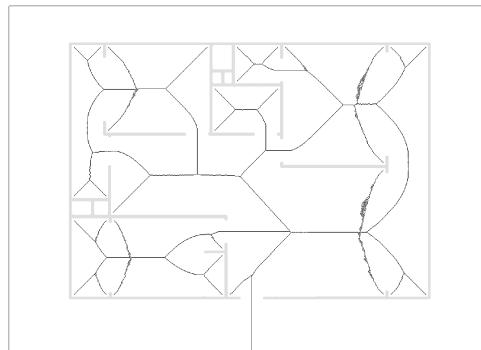
명백히 실내인 영역과 실외인 영역을 제외하면 출입문, 가구, 부속 실 등 색이나 모양으로 실내외가 구분되지 않는 공간이 남는다. 예를 들어, 단위세대 바깥쪽으로 난 현관문이 그려진 영역은 실외이지만, 안쪽으로 열리는 침실 문은 침실 안에 그려진다. 흰색이나 회색으로 그려진 가구 및 부속 실도 어디에 위치하고 어떤 공간과 연결되느냐에 따라 해당 영역의 성격이 달라진다.

이러한 영역들은 실내외가 명백한 영역 중 가장 밀접하게 연결된 공간과 같은 영역으로 분류한다. 벽으로 막힌 부분을 제외하고, 실내외가 명백한 영역에 닿아있는 인접한 공간은 여러 영역 중 가장 넓게 맞닿아있는 영역과 같은 실로 분류한다. 해당 공간을 다시 실내외가 명백한 공간으로 분류하고, 그 공간에 맞닿은 또 다른 공간을 실내외로 분류하는 과정을 더 이상 새로운 공간이 실내 외로 분류되지 않을 때까지 반복한다.

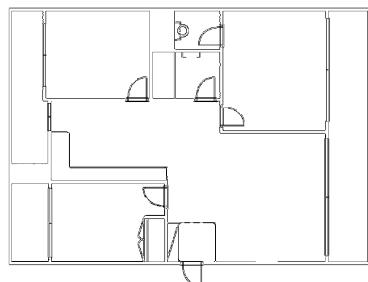
이러한 과정을 거친 후에도 평면도의 모든 부분이 실내와 실외로 구분된 것은 아니다. 예를 들어, 벽으로 둘러싸인 설비 공간은 어느 공간과도 맞닿지 않기



(가) 검은색 선 인식



(나) 실내 공간의 연결관계



(다) 실 경계 후보

그림 4.5: 단위평면 실 경계 후보 인식

때문에 실내외 어느 쪽으로도 분류되지 않는다. 이 연구에서는 단위평면의 주거공간 배치를 각 실 영역의 종합으로 정의하였기 때문에 실내에서 접근할 수는 없지만 단위세대 내부에 속하는 공간과 세대 외부는 똑같이 실내 어느 실 영역에도 속하지 않는 공간으로 취급된다. 그러나 실제로 거주자가 생활하는 공간의 배치를 분석한다는 점에서 단위세대의 각 실에서 접근이 가능한 실내 공간의 배치만 표현하여도 아파트의 공간 계획을 충분히 표현할 수 있다.

이러한 과정을 통하여, 단위평면 내부에 위치한 벽과 현관, LDK, 침실, 발코니, 화장실 영역을 인식하고, 인식된 주거공간 배치 데이터를 2차원 이미지에서 각 영역을 2차원 행렬로 표현하여 쌓은 (이미지 폭×이미지 높이×6개 영역) 크기의 3차원 행렬의 형태로 처리한다.

이와 같이 처리된 단위평면 주거공간 배치 인식 최종 결과물은 그림 4.7와 같다.

### 4.3 평면도 정규화

한국 아파트 단위평면을 주거공간 배치 분석에 적용할 수 있는 형태로 축척과 방향을 정규화 (normalization) 한다. 단위평면에서 나타나는 주거공간 배치를 딥 러닝 모형을 활용하여 분석하기 위해서는 각 공간의 크기와 전체 단위평면에서의 배치가 정확하게 표현되어야 한다. 그러나 4.1절에서 수집된 한국 아파트 평면도 이미지는 부동산 거래를 위한 보조 자료로서 사람이 볼 때 대략적인 주거공간 배치를 파악할 수 있도록 작성된 것으로, 전체 평면의 규모나 주 향의 방향 등이 일관되게 표현되지 않는다.

이 절에서는 평면도와 함께 수집된 다른 데이터와 한국 아파트의 계획 경향 등을 적용하여 수집된 평면도에 대한 축척과 방향 정규화를 수행한다.

<sup>63</sup>NAVER Corp. (2017). 네이버 부동산.

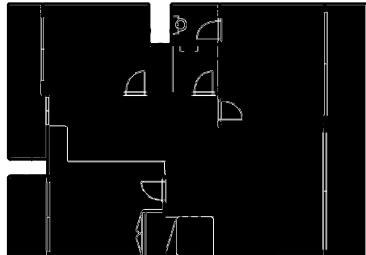
<sup>64</sup>NAVER Corp. (2017). 네이버 부동산.



(가) 평면도 원본63

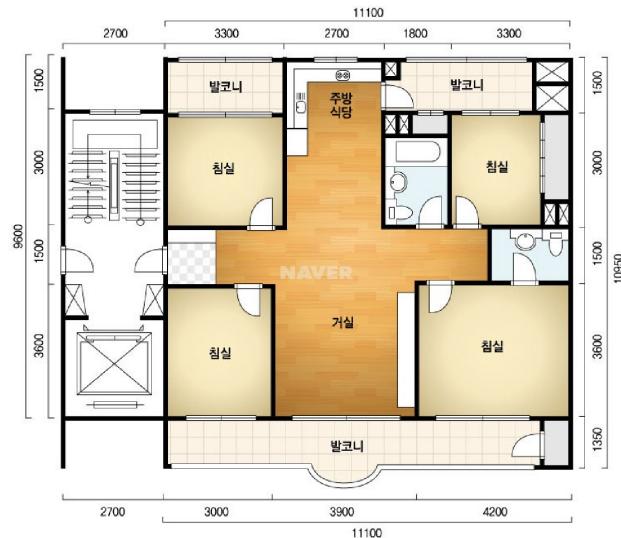


(나) 개별 영역 인식

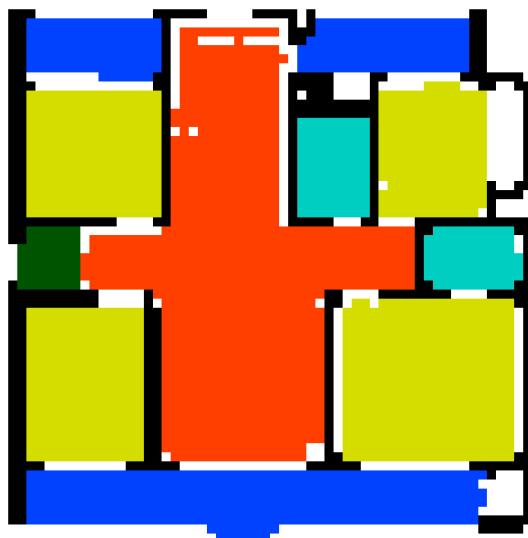


(다) 실내 영역 인식

그림 4.6: 단위평면 실내 영역 인식



(가) 평면도 원본<sup>64</sup>



(나) 주거공간 배치 인식

그림 4.7: 단위평면 주거공간 배치 인식

### 4.3.1 평면도 방향 정규화

평면도 정규화의 첫 단계는 비슷한 단위평면이 같은 방향으로 비교될 수 있도록 단위평면의 방향을 정규화한다. 유사한 배치의 단위평면에서 모든 공간이 유사한 위치에 나타나도록 하여 단위평면 분석에서 주거공간 배치의 유사성이 정확하게 인식되도록 한다.

수집된 평면도에서는 단위세대의 실제 배치와 무관하게 다양한 방향으로 표현된다. 대다수의 평면도에서 단위평면은 거실과 주 침실이 배치된 전면이 아래로 향하도록 표현된다. 남향 판상형 주동의 경우 이러한 평면도는 도면 윗쪽이 정북 방향으로 그려진 배치도와 같은 방향을 향하게 된다. 따라서 다른 방향의 판상형 아파트나 탑상형 아파트의 평면도에서도 단위세대 전면이 아래를 향하도록 그려지는 것은 자연스러운 현상일 수 있다. 그러나 모든 평면도에서 단위세대 전면이 반드시 아래쪽을 향하도록 그려진 것은 아니다. 전면이 좁고 깊은 소형 단위세대 평면 유형의 경우, 가로로 긴 평면도 이미지 안에서 단위세대가 더 크게 표현되도록 90도 회전된 형태로 표현되는 경우가 있다. 또한, 일부 평면도는 코어를 도면의 아래 부분에 배치하여, 단위세대의 여러 실이 위로 향하도록 그려지기도 한다. 이처럼 수집된 평면도는 일관된 규칙 없이 다양한 방향으로 표현되어 있기 때문에 정확한 단위평면 분석을 위하여 단위세대의 방향을 일치시켜 비교할 필요가 있다. 이 연구에서는 수집된 평면도의 대부분이 그려진 방식을 기준으로 단위평면의 주 향이 아래쪽에 오도록 정규화한다.

서로 다른 단위평면을 동일한 방향으로 통일시키기 위해서는 단위평면의 주 향을 판단하기 위한 기준이 필요하다. 판상형 주동의 단위평면에서는 주 향이 대체로 명백하게 드러난다. 편복도형 또는 계단실형 주동에서 인접한 단위 세대는 단위평면의 양 측면을 가리게 되므로, 단위평면은 전면과 후면에서만 개방될 수 있다. 전통적인 남향 주거 선호 경향에 따라 남향에 가까운 입면에

거실과 주 침실 등 주요 생활공간이 배치되어 단위평면의 전면이자 주 향의 역할이 부여된다.

그러나 모든 단위평면에서 주 향이 항상 명확한 것은 아니다. 다양한 이유로 여러 단위세대 평면도가 서로 다른 방향을 향하도록 그려지는 것과 마찬가지로, 일부 단위평면에서는 주 침실과 거실이 서로 다른 방향으로 배치되는 등 주 향을 결정하는 여러 요인이 서로 다른 방향을 나타내기 때문이다. 전면 2베이 이상 단위평면에서 대체로 거실과 주 침실이 전면에 배치되지만, 소형 평면 중에는 거실이 따로 없이 침실 2개가 전면에 배치되기도 한다. 판상형 주동에서 단위세대의 양 측면이 개방되지 않은 경우가 대부분이지만, 측면이 개방된 평면에서 전면이 아닌 측면에만 발코니가 계획되기도 한다. 탑상형 주동의 단위평면에서는 하나의 주된 향이 존재한다는 개념 자체가 도전받는다. 전면과 측면이 개방된 탑상형 평면에서는 거실과 주 침실이 남향으로 배치되어도 LDK 와 나머지 침실이 4베이까지 개방되는 동향/서향이 거주자들에게 일반적으로 인지되는 개방면이 된다. 이처럼 한국 아파트의 전형적인 단위평면으로 대상을 한정하더라도 모든 단위평면에서 일관된 주 향을 정의하는 것은 어렵다.

이 연구에서는 이러한 한계를 고려하여 다양한 단위평면의 공간 배치를 최대한 일관되게 비교할 수 있는 최소한의 기준을 수립한다. 한국 아파트에서 나타나는 다양한 단위평면에서 일관된 주 향을 정의하기 위해서는 주 향을 결정하는 가장 본질적인 특성을 가려낼 필요가 있다.

앞에서도 언급하였듯이, 한국 아파트에서 주 향을 정의하는 가장 중요한 특징은 남쪽을 향한 단위평면 전면에 주요 생활공간이 배치된다는 점이다. 건축법 시행령에서도 일조의 확보 대상으로 “주된 개구부”를 “거실과 주된 침실이 있는 부분의 개구부”로 정의한다. 따라서 거실과 주 침실이 남향으로 계획되는 경우 그 방향이 단위평면의 주 향이 될 수 있다.

그러나 이러한 특성은 한국 아파트 평면 대부분에서 일반적으로 성립하지만,

탑상형 주동의 단위평면에는 잘 들어맞지 않는다. 이 기준에 따르면 그림 4.8에 표현된 탑상형 주동의 서로 인접한 두 면이 개방된 평면이 남향으로 배치된 주 침실을 따라 전면 2베이 평면으로 해석될 수 있다. 그러나 단위평면 내부의 공간 배치를 고려하면, 거실이 더 넓게 개방되고 나머지 모든 생활공간이 배치된 쪽을 주 향으로 보아 전면 4베이 평면으로 해석하는 것이 더 적합하다. 선행연구에서도 이와 유사한 탑상형 평면을 전면 4-5베이 평면으로 해석하고 있다.<sup>65</sup>



그림 4.8: 전면과 측면이 개방된 탑상형 주동 평면 사례

이 연구에서는 한 면만 개방된 1베이 원룸부터 3면이 개방된 탑상형 평면까지 다양한 평면에 적용할 수 있는 기준으로, 주 향을 주요 생활공간의 개구부가 가장 많이 배치된 방향으로 정의한다. 이러한 정의는 전면만 개방된 중복도형 도시형생활주택의 평면이나, 전면과 후면이 개방된 편복도형 혹은 계단실형 주동의 평면, 3면이 개방된 탑상형 평면 등 다양한 조건의 단위평면에 일관된

<sup>65</sup>Byun, Choi. (2016). A Typology of Korean Housing Units: In Search of Spatial Configuration. Journal of Asian Architecture and Building Engineering, 15(1), 41-48.; 윤효진. (2019). 아파트의 주동형태 및 확장형 발코니에 따른 단위세대 평면계획 변화특성. 한국퍼실리티매니지먼트학회지, 14(1), 61-69.; 이상진, 박소현. (2019). 부산 아파트 단지의 주동 평면형태의 변화 특성에 관한 연구. 대한건축학회 논문집-계획계, 35(8), 3-14.

적용이 가능하다.

그러나 4장에서 수집된 평면도에는 개구부가 정확히 표현되지 않은 경우가 많기 때문에, 단위평면에서 개구부가 배치된 방향을 직접적으로 판별하는 것은 불가능하다. 따라서 이 연구에서는 수집된 평면도에서 파악할 수 있는 정보로 주요 생활공간이 가장 많이 배치된 개방면을 추론하고 그에 따라 주 향을 결정 한다.

건축법 시행령이 1986년과 1988년 두 차례에 걸쳐 개정되면서 공동주택의 발코니 면적이 바닥면적 산정에서 제외된 이후, 한국 아파트에서는 단위평면 전면 전체에 걸친 발코니 계획이 일반화된다. 따라서, 수집된 평면도에도 대부분의 경우 발코니가 표현되어 있으며, 그 배치를 통하여 개방면의 방향을 확인할 수 있다. 예를 들어, 침실이나 거실의 아래쪽에 좌우로 긴 발코니가 있다면 해당 실은 아래 방향에서 외기와 면하고 있다는 것을 알 수 있다. 만약 주요 생활공간에 계획된 여러 발코니에서 같은 배치가 나타난다면, 단위평면의 주 향이 아래쪽으로 오도록 평면도가 그려졌다고 판단할 수 있다.

발코니 배치에 기반한 개방면 추론을 적용한 단위평면의 주 향 판단은 다음과 같은 순서로 진행된다. 먼저 평면도에서 인식된 발코니 영역을 공간적 연결을 기준으로 나누어 개방면 추론의 대상으로 설정한다. 개방면 추론의 대상이 되는 발코니 영역이 반드시 하나의 실에 계획된 개별 발코니를 의미하는 것은 아니다. 예를 들어, 여러 실 앞에 일렬로 배치된 발코니가 하나로 연결되는 경우처럼, 평면도에 따라 여러 발코니가 하나의 발코니 영역으로 나타나기도 한다. 불완전한 평면도에서도 정확하게 주 향을 판단할 수 있는 알고리즘이 될 수 있도록, 이후 단계에서는 여러 발코니가 하나의 발코니 영역으로 인식되어 도 결과가 달라지지 않도록 한다.

다음으로, 각 발코니 영역이 배치된 방향을 발코니 영역의 세장비와 단위평면 안에서의 배치에 기반하여 판단한다. 각 발코니 영역이 배치된 방향을 판단하

는 이유는 가장 많은 발코니가 배치된 주 향을 판단하기 위한 것이다. 대다수의 평면도에서 단위평면의 주 향이 평면도의 상하좌우 4방향 중 하나를 향하고 있다고 가정하고, 따라서 주 향을 향한 발코니 또한 같은 방향으로 배치되었다고 가정한다.

각 발코니 영역이 주 향이 될 수 있는 4방향 중 어느 방향에 해당될 수 있는지를 판단한다. 발코니 영역의 수평 방향 폭과 수직 방향 높이를 비교하여 세장비를 측정하여, 폭이 높이보다 클 경우, 평면도의 상하 방향 개방면에 배치된 가로로 긴 발코니로 판단하고, 그 반대의 경우 좌우 방향으로 배치된 세로로 긴 발코니로 판단한다. 세장비를 통하여 압축된 상하 또는 좌우 두 방향 중에서 발코니 영역의 향이 어느 쪽인지는 단위평면 안에서 발코니와 인접 공간의 배치를 통해 파악할 수 있다. 모든 발코니는 다른 주거 공간에 부속되어 계획되므로, 발코니 영역의 양쪽 중 한 쪽에는 다른 공간이 인접하게 된다. 즉, 발코니 길이에 비하여 좁은 폭이 나타나는 두 방향 중 다른 주거 공간이 인접하여 배치되지 않은 쪽이 발코니 영역이 향한 단위평면 외부의 개방면이 된다. 발코니라는 공간의 특성상 양쪽에 실내 공간이 모두 배치될 수 없기 때문에 다른 주거 공간이 전혀 배치되지 않은 쪽이 반드시 존재하여야 하고, 그 방향을 발코니가 향한 개방면으로 판단할 수 있다. 그러나 평면도의 오류 가능성을 고려하여, 양쪽 중에서 실내 공간이 더 적게 배치된 쪽을 외부로, 즉 해당 발코니 영역의 배치 방향으로 판단한다.

마지막으로, 각 발코니 영역의 배치 방향을 종합하여 단위평면의 주 향을 판단한다. 발코니의 개방면에 기반한 주 향 판단은 연속된 여러 발코니가 하나의 발코니 영역으로 인식된 경우에도 그 결과가 달라지지 않아야 한다. 따라서 발코니가 몇 개의 영역으로 인식되는가와 관계없이 달라지지 않는 특성을 기준으로 발코니 배치 방향을 종합한다. 이 연구에서는 발코니 영역의 길이를 발코니 배치 방향의 종합을 위한 가중치로 삼는다. 구체적으로, 각 발코니 영역을 평면도의 상하좌우 4방향에서 본 길이를 기준으로 발코니가 배치된 방향의

길이를 각 방향마다 집계한다.

주요 생활공간과 부속된 발코니가 한 방향에 집중적으로 배치되는 경우, 단위 평면의 주 향은 해당 방향으로 쉽게 결정된다. 한 방향에 배치된 발코니가 다른 방향보다 훨씬 많거나, 아예 모든 발코니가 한 방향에만 배치된 경우, 발코니가 가장 많이 배치된 방향을 발코니가 계획된 주요 생활공간이 가장 많이 배치된 방향으로, 즉 단위평면의 주 향으로 판단할 수 있다.

그러나 여러 단위세대가 일렬로 배치되는 판상형 주동에서는, 단위평면 전면과 후면의 발코니 면적이 같거나 유사한 규모로 나타날 수 있다. 이러한 경우에 발코니 면적의 미세한 차이는 주요 생활공간이 계획되었음을 나타내는 기준이 될 수 없다. 따라서 여러 방향의 발코니 길이 합계가 큰 차이를 보이지 않는 경우에는, 주요 생활공간이 여러 개방면 중 어느 쪽에 면하여 배치되었는가에 따라 주 향을 판단할 필요가 있다.

이 연구에서는 주 향의 판단 기준이 되는 주요 생활공간을 일조의 확보 대상 중 주 침실이 되도록 한다. 한국 아파트에서 주 침실이 배치된 방향이 주 향의 반대편, 즉 북향에 해당하지 않는다는 판단을 전제하고, 서로 반대되는 방향으로 발코니가 배치된 경우, 주 침실에 가까운 쪽만 주 향 후보로 남기고 먼 쪽은 후보에서 배제한다. 구체적으로는, 평면도의 침실 영역 안에서 벽에서 가장 먼 지점을 가장 큰 침실의 중심, 즉 주 침실의 중심으로 인식하여, 발코니가 배치된 반대 방향 개방면 중에서 이 지점과 더 먼 쪽을 후보에서 배제한다.

양 측면이 코어 또는 인접 단위세대로 막힌 계단실형 주동의 단위평면에서는 이러한 과정을 통하여 한 방향만 남게 되고, 이 방향이 주 향이 된다. 그러나 각각 방향으로 배치된 두 방향으로 발코니가 배치된 평면에서는, 위의 과정을 거치고 나서도 두 방향 모두 주 향 후보로 남게 된다. 이러한 경우 발코니가 더 많이 배치된 방향을 주 향으로 한다. 이렇게 인식된 주 향이 아래 방향으로 통일되도록 단위평면을 정렬한다.

수집된 평면도는 주 향의 방향 뿐만 아니라 코어를 중심으로 대칭적으로 나타나는 반전된 단위평면에 대한 정규화도 필요로 한다. 네이버 부동산에서는 대칭적인 단위평면을 모두 같은 평면 유형으로 분류하며, 평면도에는 서로 반전된 두 단위평면 중 한 쪽이 일관되지 않게 표현된다. 따라서 유사한 단위평면이 반전되어 표현된 경우에도 주거공간 배치의 유사성을 정확하게 인식할 수 있도록 단위평면과 공용 공간의 방향을 정규화한다. 이를 위하여, 단위평면과 공용 공간을 연결하는 공간인 현관을 기준으로 삼아, 주 향을 축으로 현관이 항상 한 쪽으로 향하도록 단위평면을 반전시킨다. 반전된 단위평면 중 반드시 한 쪽을 선택해야만 하는 이유는 없으므로, 수집된 평면도 다수의 배치를 따라 현관이 단위평면의 왼쪽에 위치하도록 한다.

이러한 평면도 방향 정규화 과정을 통하여, 모든 단위평면에서 일관되게 주 향이 아래쪽으로 향하고 현관이 단위평면의 왼쪽에 위치하도록 정규화한다.

#### 4.3.2 평면도 축척 정규화

평면도 정규화의 다음 단계로 다양한 단위평면의 축척을 딥 러닝 분석에 적합한 값으로 일치시키는 것이다. 단위평면 속 주거 공간 배치를 비교하기 위해서는 개별 공간의 크기도 중요한 정보이므로 단위평면의 규모를 정확하게 반영하는 것이 중요하다.

수집된 평면도에서 단위평면은 다양한 축척으로 표현된다. 단위평면은 일정한 크기의 이미지를 가득 채우도록 배치되어 축척이 일정하게 정해지지 않고, 표현되는 공간의 면적에 따라 축척이 달라지게 된다. 동일한 규모의 단위평면 사이에서도 평면의 세장비에 따라, 코어 및 인접 단위세대의 표현 여부에 따라 표현되는 공간의 크기가 영향을 받으므로 축척이 달라질 수 있다.

일부 평면도에는 치수가 표기되어 있기도 하지만, 대다수의 평면도에서는 축 척과 치수가 표기되어 있지 않다. 따라서 수집된 평면도에 포함된 정보만으

로는 모든 평면도에서 정확한 축척을 파악할 수 없다. 이 연구에서는 네이버 부동산에서 평면도와 함께 공개하는 전용면적 값을 활용하여 단위평면의 축척을 정규화한다. 4.2.2절에서 인식된 단위평면의 실내 영역 중에서 발코니를 제외한 나머지 실내 공간 영역의 합과 전용면적을 비교하여 평면도의 축척을 산출한다.

각 평면도의 축척을 파악한 이후에는 평면도 이미지의 크기를 조절하여 딥러닝 분석에 적합한 새로운 축척으로 단위평면의 축척을 정규화한다. 새로운 축척은 한국 아파트 단위세대의 규모와 딥 러닝 모형의 요구조건을 고려하여 결정된다. 이 연구에서 구축하는 한국 아파트 데이터셋도 다양한 딥 러닝 모형에 적용할 수 있도록 한국 아파트의 단위평면을 이러한 이미지 크기 안에 담을 수 있는 축척을 설정한다.

기계학습 및 딥 러닝 연구에서 많이 활용된 MNIST 데이터셋은 가로 및 세로가 28 픽셀인 이미지로 이루어져 있으며, 따라서 MNIST를 대상으로 설계된 모형은 크기가 28 픽셀인 이미지의 처리에 최적화되어 있다. 그 이후에도 다양한 데이터셋과 딥 러닝 모형에서 28픽셀에서 크기를 2배씩 증가시킨 56, 112, 224 픽셀 크기의 이미지를 입력값으로 받는다. CNN 모형의 내부 구조에서는 이미지 크기를 절반으로 줄이는 계층이 자주 활용되므로, 크기를 2배씩 증가시킨 이미지는 다양한 딥 러닝 모형에 쉽게 적용할 수 있다. 이러한 이미지 크기에 담기는 단위평면의 크기는 수집된 평면도를 통하여 파악할 수 있다. 4.3절의 알고리즘을 적용하여 방향과 축척이 정규화된 평면도를 대상으로, 여백을 제외하고 단위평면 영역의 폭과 깊이를 분석한 결과는 그림 4.9과 같다. 이에 따르면, 한국 아파트의 단위평면은 폭과 깊이 모두에서 약 12 미터인 경우가 가장 많이 나타나며, 가장 큰 단위평면도 폭과 깊이 모두에서 그 두 배인 24 미터 이하에 해당한다. 24 미터를 56, 112, 224 픽셀 안에 대응시키기 위해서는, 미터당 2.3, 4.7, 9.3 픽셀의 축척이 필요하다.

이 연구에서는 원본 평면도 이미지의 정확도를 고려하여 미터당 5 픽셀 (5 px/

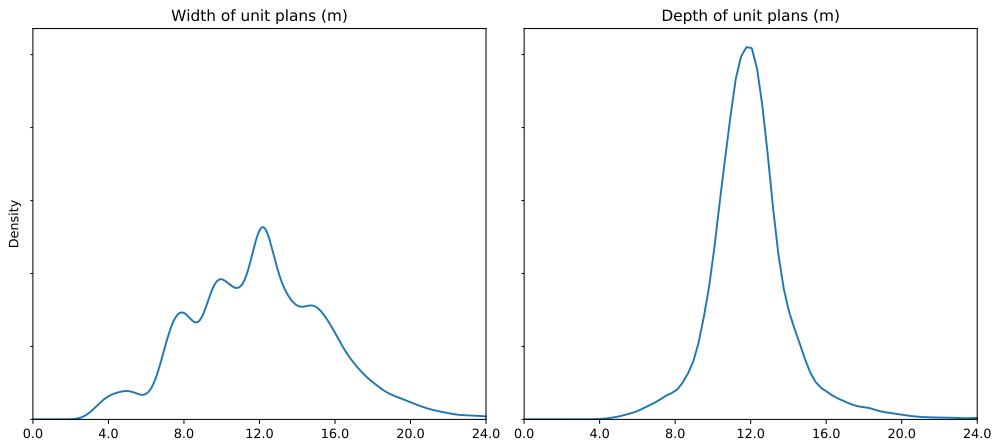


그림 4.9: 단위평면의 폭 및 깊이 분포

m)의 축척을 정규화에 적용한다. 이 경우 112 픽셀이 22.4 미터에 해당하여 한국 아파트에서 나타나는 가장 큰 단위평면 전체를 딥 러닝 모형의 입력값으로 표현할 수 있다. 최종적으로 한국 아파트 단위평면 데이터셋에서 주거공간 배치 데이터는 가로 세로 112 픽셀 크기의 이미지에 6개 주거 공간의 정보를 담는 ( $112 \times 112 \times 6$ ) 크기의 행렬로 표현된다.

## 4.4 분석 영역 설정

딥 러닝 분석을 통하여 다양한 크기의 단위평면을 비교하기 위한 단위평면의 분석 영역을 설정한다.

딥 러닝 모형은 입력값의 모든 영역에서 의미있는 정보로 차 있어야 학습이 정상적으로 이루어져 유의미한 분석이 가능하다. 이 연구의 단위평면 데이터셋에서는 다양한 크기의 단위평면을 일관된 축척으로 표현하기 때문에, 단위평면의 규모에 따라 주거공간 배치 데이터도 서로 다른 크기로 표현된다. 만약 가장 큰 단위평면에 맞추어진 주거공간 배치 데이터 전체를 딥 러닝 모형의 입력값으로 적용한다면, 대다수의 단위평면에서는 단위평면 외부의 무의미한

공간이 가장 큰 비중을 차지하게 된다. 이 경우 딥 러닝 모형은 단위평면 내부의 주거공간 배치에 대한 정상적인 학습이 불가능하다.

따라서, 평균적인 크기의 단위평면에서 실내의 주거공간 배치는 모두 담아내면서도 여백을 남기지 않는 크기의 분석 영역을 설정할 필요가 있다. 이러한 분석 영역은 단위평면에서 주거공간 배치의 유사성과 차이가 가장 잘 나타나는 영역에 설정되어야 한다. 다양한 조건의 단위평면에서 분석 영역이 일관된 위치에 설정되고, 그 안의 주거공간 배치에서 잠재적인 유사성이 드러나도록 한다.

4.3.2절에서 설정했던 주거공간 배치 데이터의 크기와 마찬가지로, 분석 영역의 크기 또한 딥 러닝 모형의 요구조건에 맞추어 28 픽셀을 2배씩 증가시킨 크기에 맞추어 설정한다. 그림 4.9에서 나타나듯이, 한국 아파트 단위평면의 폭과 깊이 모두에서는 약 12 미터에 집중되어 있다. 이를 주거공간 배치 데이터셋에 적용된 5 px/m의 축척으로 환산하면 이는 약 60 픽셀에 해당한다. 28 픽셀의 2배인 56 픽셀은 이에 가장 가까운 크기로 같은 축척에서 11.2 미터를 표현할 수 있고, 평균적인 크기의 한국 아파트 단위평면에 대하여 최대한 많은 정보를 담아내면서도 분석 영역을 실내 공간으로 가득 채울 수 있다.

다음으로, 서로 다른 규모의 단위평면 사이에서 나타나는 주거 공간 배치의 유사성을 분석하기 위하여, 단위평면 안에서 분석 영역의 위치를 설정한다. 이 연구에서는 공간의 실제 규모와 단위평면의 전반적 배치를 주거공간 배치 데이터에 반영한다. 따라서, 단위세대의 규모에 따라 단위평면의 크기는 고정된 크기의 분석 영역보다 크거나 작게 나타날 수 있다. 분석 영역의 위치를 설정하는 기준은 단위평면이 분석 영역보다 작거나 큰 경우에도 일관되게 적용할 수 있어야 한다. 또한, 서로 규모가 다른 두 평면을 비교하는 경우 그 안에서 공간 배치가 가장 유사한 같은 면적의 영역이 분석 대상으로 설정될 수 있도록 한다.

이를 위하여, 분석 영역의 기준점을 인접한 다른 단위세대와 공용 공간에서 가장 먼 지점인 주 향 입면의 중앙으로 설정하고, 이를 분석 영역의 하단에 위치시킨다. 이렇게 분석 영역을 설정하였을 때, 주거공간 배치 행렬의 하단 중앙부에는 모든 단위평면에서 항상 주 향에 면한 주요 주거 공간이 배치된다. 이러한 분석 영역의 설정을 통하여 다양한 규모와 배치를 보이는 단위평면을 주 향 입면에서 전개되는 실내 공간 계획을 기준으로 비교할 수 있다.

## 4.5 평면유형 연관 데이터 처리

네이버 부동산에서 수집된 자료 중에는 평면도 외에도 평면유형과 연관된 정보가 포함된다. 이러한 정보는 한국 아파트의 진화과정을 분석하기 위하여 단위평면과 함께 딥 러닝 모형에 적용될 수 있다. 평면도와 마찬가지로, 이러한 데이터도 딥 러닝 분석에 적합한 형태로 정규화하여 한국 아파트 데이터셋에 함께 수록한다.

### 4.5.1 준공연도 정규화

아파트 계획의 시간에 따른 변화를 분석하기 위하여 단위평면이 계획된 시기에 대한 정보가 필요하다. 이 연구에서는 네이버 부동산에서 제공하는 준공연도를 그 기준으로 한다. 대다수의 선행연구에서, 아파트의 계획 시기는 10년 단위 연대를 전후반으로 나눈 5년 단위 구간을 기준으로 구분한다.<sup>66</sup> 대규모

<sup>66</sup>Byun, Choi. (2016). A Typology of Korean Housing Units: In Search of Spatial Configuration. Journal of Asian Architecture and Building Engineering, 15(1), 41-48.; 박인석 등. (2014). 전용면적 산정기준 변화와 발코니 용도변환 허용이 아파트 단위주거 평면설계에 미친 영향: 전용면적 60m<sup>2</sup> 와 85m<sup>2</sup> 평면의 실별 규모 변화를 중심으로. 한국주거학회논문집, 25(2), 27-36.; 이상진, 박소현. (2019). 부산 아파트 단지의 주동 평면 형태의 변화 특성에 관한 연구. 대한건축학회 논문집-계획계, 35(8), 3-14.; 최병숙, 박정아. (2009). 여성관련 공간을 중심으로 본 서울지역 아파트의 공간구조 변화. 한국주거학회논문집, 20(4), 39-47.; 최재필. (1996). 공간구문론을 사용한 국내 아파트 단위주호 평면의 시계열적 분석-수도권 4LDK 아파트 단위주호 평면계획을 중심으로. 대한건축학회 논문집,

아파트 건설이나 계획과 관련된 법규 변화 등을 반영하여 구간의 범위를 1년 정도 옮기는 경우도 있으나, 이러한 구간 범위 조정이 모든 연구에서 일관되게 나타나지는 않는다. 이 연구에서도 이 장에서 구축하는 데이터셋을 활용한 분석 결과를 선행연구와 비교할 수 있도록, 준공연도를 5년 단위 구간으로 정규화한다.

네이버 평면도에서 평면도와 함께 수집된 준공연도는 1969년부터 2019년까지 51년의 범위에 걸쳐 있다. 단순하게 10년 단위 연대를 전후반으로 나누어 구분하면 1969년은 단독으로 1960년대 후반에 속하게 되나, 1969년에 준공된 사례는 2개 단지, 4개 평면유형에 불과하여 해당 시기에 준공된 아파트를 충분히 대표하지 못한다. 따라서 1969년 사례를 1970년대 전반에 포함하여 시기 구분을 설정한다. 최종적인 시기 구분과 해당 시기의 아파트 계획 관련 상황은 표 4.1와 같다.

표 4.1: 시기 구분 정의

코드	시기	준공연도	시기적 상황
0	1970년대 전반	1969–1974	여의도 시범아파트 (1971)
1	1970년대 후반	1975–1979	강남개발, 아파트 지구 (1975)
2	1980년대 전반	1980–1984	2차 석유 파동 (1978–1981)
3	1980년대 후반	1985–1989	목동 신시가지 (1983), 3저 호황 (1986–1989), 발코니 외벽 기준 면적 산입 제외 (1988)

12(7), 15–27.; 최재필. (1996). 한국 현대 사회의 주생활양식의 변화–수도권 3LDK 아파트 주호 평면 계획의 변천을 중심으로. 대한건축학회 논문집, 12(9), 3–12.

코드	시기	준공연도	시기적 상황
4	1990년대 전반	1990–1994	수도권 1기 신도시 (1988–), 발코니 창호 설치 허용 (1993)
5	1990년대 후반	1995–1999	외환위기 (1997), 분양가 자율화 (1999)
6	2000년대 전반	2000–2004	탑상형 아파트 도입 (2002), 발코니 창호 설치 의무화 (서울시, 2003)
7	2000년대 후반	2005–2009	발코니 확장 허용 (2006), 분양가 상한제 (2007), 뉴타운 (2002)
8	2010년대 전반	2010–2014	서브프라임 모기지 금융위기 (2008), 전국 주택보급률 100% 초과 (2008)
9	2010년대 후반	2015–2019	주택시장 안정대책 (2017), 도시재생 뉴딜 (2017)

#### 4.5.2 지역 정규화

아파트 계획의 지역별 차이를 분석하기 위하여 단위평면과 함께 해당 사례가 위치한 지역에 대한 정보가 필요하다. 평면유형의 지역은 범주형 변수에 속한다. 범주형 데이터는 그 자체로는 특별한 처리가 필요하지 않으나, 딥러닝 모형이 범주형 변수 사이의 차이를 학습하기 위해서는 각 범주에 충분한

사례가 있어야 하며, 그 수가 균형을 이루어야 한다. 그렇지 못할 경우 사례 수가 부족한 범주에 대한 정보가 학습되지 않고 지역에 따른 차이를 정확하게 분석할 수 없게 된다.

한국의 수도권 집중은 아파트의 분포에서도 나타난다. 만약 행정구역 체계에 따라 모든 행정구역을 별개의 범주로 분류한다면 인구 규모와 아파트 사례 수가 적은 지방의 아파트에 대한 학습이 제대로 이루어질 수 없다. 따라서 단순히 같은 수준의 행정구역을 기준으로 범주화하는 것이 아니라 평면유형 사례 수를 고려하여 범주화하여야 한다.

한편, 반대로 사례 수의 균형만을 고려하여 범주화를 진행한다면 전체 사례의 약 1/3씩을 차지하는 서울과 경기는 개별 범주가 되지만, 나머지 전체 사례는 하나의 범주로 묶어야 비슷한 규모의 범주가 될 수 있다. 이 경우 수도권을 제외한 나머지 지역의 차이를 분석할 수 없게 된다.

아파트 계획 지역의 정규화는 지역별 비교에 활용하는 것을 목적으로, 각 지역의 특색을 살리면서도 각 권역의 사례 수가 균형 있게 배분되도록 범주화한다. 이를 위하여 시도 단위를 기준으로, 개별 시도에 계획된 평면유형 수와 각 시도 사이의 관계를 고려하면서 인접한 시도를 묶어 권역을 설정한다. 이러한 권역 설정을 통하여 평면유형 분포의 불균형을 완화하면서 지역별 차이를 학습할 수 있다.

충청와 전라는 평면유형 사례 수가 가장 적은 편에 속한다. 따라서 2개 도와 그 안에 위치한 자치시를 묶어 하나의 권역으로 설정한다. 이 경우 충청 권역은 약 4천 개, 전라 권역은 약 2천 개 평면유형을 포함하게 된다. 이를 기준으로 다른 시도에 대한 권역을 설정하면, 서울, 경기 권역과 함께 약 3천 개인 인천이 독립된 권역으로, 경상도는 남북으로 나누어 부산과 울산을 포함한 경남 권역과 대구를 포함한 경북 권역으로, 다른 시도와 성격이 다른 강원과 제주는 별개 권역이 된다. 이러한 정규화에 따라 전국을 총 9개 권역으로 구분한다. 한국

아파트 데이터셋에서 평면유형의 지역을 나타내는 권역코드는 표 4.2와 같다.

표 4.2: 시도별 권역 정의

권역코드	해당 지역	평면유형 수
0	서울	15,125
1	경기	14,444
2	인천	3,288
3	강원	1,107
4	대전, 세종, 충남, 충북	4,226
5	부산, 울산, 경남	6,866
6	대구, 경북	3,242
7	광주, 전남, 전북	2,075
8	제주	167

### 4.5.3 전용면적 정규화

면적은 서로 다른 두 값 사이에 비율 관계가 성립하는 비율 변수이다. 예를 들어, 120 제곱미터는 60 제곱미터보다 두 배 넓은 면적인 관계가 성립한다. 비율 변수에서는 실제로 양적 없음을 의미하는 절대영점이 존재한다. 면적이 0이라는 것은 실제로 면적이 존재하지 않음을 뜻한다. 이러한 변수를 취급할 때는 정규화 과정에서 변수에 존재하는 비율 관계가 유지될 수 있도록 유의할 필요가 있다. 비율 변수에 로그를 취하는 경우, 이러한 비율 관계를 직접 등간 변수로 표현할 수 있고, 절대영점은 무한대로 옮겨져 무시할 수 있게 된다. 따라서 로그를 취한 후 데이터에서 나타나는 값의 범위를 자유롭게 옮길 수

있다.

딥 러닝에서 사용하는 신경망 모형은 각 계층에서 출력된 값을 정규화하는 과정을 거친다. 이러한 구조 때문에 연속형 변수의 경우 입력값이 0을 중심으로  $\pm 1$  정도의 편차를 가질 때 가장 자연스럽게 학습이 가능하다. 따라서 준공연도와 전용면적 등과 같은 연속형 변수는 그 값의 범위가 -1과 1 사이에 위치하도록 변환하여 정규화할 필요가 있다.

한국 아파트의 전용면적은 국민주택 규모의 상한인 85 제곱미터보다 약간 작은 사례가 가장 많이 나타난다. 소형주택도 마찬가지로 상한인 60 제곱미터에 집중되어 나타나며, 반대로 제도적인 기준이 따로 없는 대형 평면에서는 매우 넓은 범위의 전용면적이 나타난다. (그림 4.10) 따라서 정규화된 값이 0을 중심으로 나타나도록 하려면 가장 많은 사례가 집중된 국민주택 규모 85 제곱미터가 0으로 옮겨지도록 하여야 한다. 또한, 정규화된 데이터 안에서 편차가 1로 나타나려면 그 다음으로 사례가 많은 소형주택 60 제곱미터가 0에서 1만큼 떨어진 -1 또는 1의 값을 갖도록 옮겨져야 하며, 대형 주택도 평균적으로는 1의 편차를 갖도록 하여야 한다.

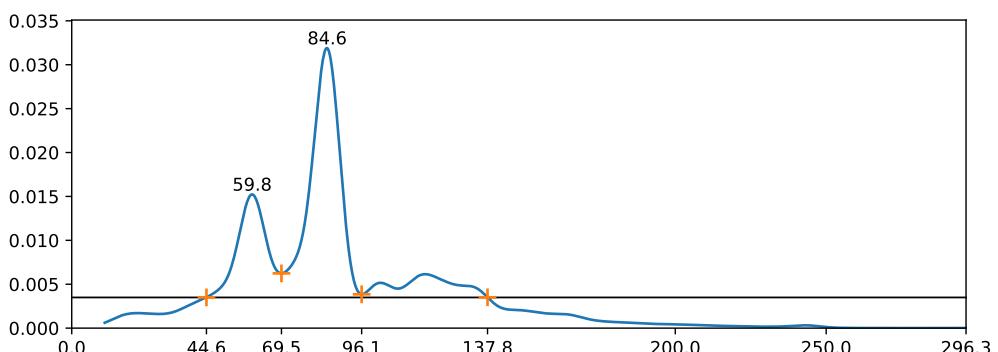


그림 4.10: 전용면적 분포

이러한 점들을 고려하여, 다음과 같이 전용면적의 정규화를 적용한다. 먼저 85 제곱미터와 60 제곱미터의 간격이 1이 되도록 로그를 취한다. 로그의 밑을

$85m^2$ 와  $60m^2$ 의 비율 (약 1.417)로 설정하면, 로그를 취했을 때 해당 비율이 1의 차이로 옮겨지게 된다. 다음으로, 85 제곱미터가 0으로 옮겨지도록 전용면적에 로그를 취한 값에서 85 제곱미터에 로그를 취한 값 (약 12.755)을 뺀다. 이 때 60 제곱미터는 정확히 -1로 옮겨지고, 대형 평면의 전용면적 범위 안에 속하는 약 120 제곱미터가 1의 값을 갖게 된다.

이렇게 하면 거의 모든 사례가 0 또는 0에서 1만큼 떨어진 값을 중심으로 분포하게 되어 딥 러닝에서 요구하는 입력값의 조건에 부합한다. 원래의 전용면적을 area라고 할 때, 정규화된 전용면적 norm\_area는 다음과 같이 정의된다.

$$\text{norm\_area} = \frac{\log(\text{area}) - \log(85)}{\log(85/60)} \quad (4.1)$$

#### 4.5.4 침실 및 화장실 수

침실 수와 화장실 수는 절대영점이 있고 비율 관계가 성립하는 변수이다 (화장실 2개는 화장실 1개의 두 배). 그러나 현실적으로 한국 아파트의 단위평면에서 침실 수와 화장실 수는 다양하게 나타나지 않는다. 대부분의 단위평면에서 침실 수는 2-4개, 화장실 수는 1-2개의 값을 가진다. 이렇게 몇 가지 범주로 변수를 표현할 수 있는 경우, 해당 변수를 비율 변수가 아닌 명목 변수로 취급하여 양적 관계보다는 대등한 범주 사이의 비교에 활용할 수 있다.

많은 선행연구에서도 침실 수가 증가함에 따라 나타나는 선형적인 관계보다는 침실이 하나씩 증가하거나 감소함에 따른 평면 계획의 변화를 살핀다. 이러한 연구에는 침실 수와 화장실 수에 대한 전처리가 필요하지 않으며, 딥 러닝 모형에서 해당 데이터를 범주형 변수로 취급하기만 하면 된다. 따라서 침실 및 화장실의 개수는 별도 처리 없이 원래 값을 데이터셋에 포함한다.

## 4.6 단위평면 데이터셋 구축

위에서 구축된 단위평면 관련 자료를 종합하여 딥 러닝 분석에 적용할 수 있는 형식의 데이터셋을 구축한다. 평면도에서 단위세대의 주거공간을 인식하고 정규화한 단위평면 주거공간 배치 데이터와, 한국 아파트의 진화과정을 분석하는 데 함께 필요한 준공연도, 전용면적, 지역 등 계획 관련 정보를 정리하고 데이터셋으로 구축한다. 데이터셋에 포함된 단위평면 관련 자료 항목과 그 정의는 표 4.3와 같다.

표 4.3: 한국 아파트 단위평면 데이터셋 구성

변수명	형식	정의
floorplan	3차원 배열	주거공간 배치
plan_id	텍스트	평면유형 ID
year	정수	준공 시기 (표 4.1)
sido	정수	시도별 권역 (표 4.2)
norm_area	실수	로그 정규화한 전용면적 (식 4.1)
num_rooms	정수	침실 수
num_baths	정수	화장실 수

데이터셋은 딥 러닝 분석 플랫폼인 TensorFlow에 활용될 수 있는 TFRecords 파일 형식으로 구축된다. 분석 모형에 따라 전체 크기 평면과 여백이 없는 주거 공간 배치를 분석할 수 있도록 112 픽셀과 56 픽셀 크기의 두 가지 데이터셋을 각각 구축한다. 또한, 기계학습 모형은 데이터를 보고 학습한 성과를 정확하게 평가하기 위하여 학습 과정에서 문제와 답을 볼 수 있는 훈련 (training) 데이

터셋과 검증 과정에서 문제만 보고 답을 맞혀야 하는 시험 (test) 데이터셋을 필요로 한다. 따라서 전체 데이터셋과 더하여, 전체 사례의 약 80%를 포함하는 훈련 데이터셋과 약 20%의 시험 데이터셋도 함께 구축한다. 동일한 아파트 단지에서는 동일한 설계 주체, 시점, 지역 등이 겹치므로 단위평면 계획에서 유사성이 더 클 수 있다. 시험 데이터셋에서 제시하는 단위평면과 같은 단지의 단위평면이 훈련 데이터셋에도 존재한다면 한국 아파트 단위평면을 학습했는지 정확하게 검증이 이루어지지 않을 수 있다. 따라서 훈련 데이터셋과 시험 데이터셋을 아파트 단지를 기준으로 나누어, 같은 단지의 단위평면은 모두 훈련 데이터셋이나 시험 데이터셋 한 쪽에만 속하도록 한다.

이 장에서 구축한 단위평면 데이터셋은 공개된 평면도를 활용한 한국 아파트 전수조사에 활용할 수 있다. 전국 아파트 단위평면 수만 건을 분석하는 것은 선행연구에서는 시도된 적 없는 작업으로, 3장의 분석 방법론과 이 장의 데이터셋 구축을 함께 적용하여 가능해졌다. 다음 장에서는 이 연구의 방법론을 적용하여 한국 아파트의 진화과정을 분석하고, 이를 통해 이 연구의 방법론을 검증하고자 한다.

# 제 5 장

## 한국 아파트 단위평면 분석

### 5.1 시기별 주거공간 배치 분석 모형

이 장에서는 4장에서 구축한 한국 아파트 단위평면 데이터셋을 활용하여 시기에 따른 단위평면의 주거공간 배치 변화를 분석한다. 이를 위하여, 먼저 시기별 주거공간 배치를 학습하고 각 시기별 단위평면을 분류할 수 있는 딥 러닝 모형을 구축하였다. 구체적으로, 이 연구에 적용하는 딥 러닝 모형은 3.2절에서 선정한 VGG-GAP 모형을 단위평면 데이터셋에 맞게 수정하고 새로운 상태에서 단위평면 데이터셋을 학습시킨 것이다.

Zhou 등의 VGG-GAP 모형은 원래 VGGNet 모형과 마찬가지로 RGB 이미지를  $224 \times 224 \times 3$  크기의 행렬로 입력받아, 사진 속 대상을 1000개 클래스 중 하나로 분류한다.<sup>67</sup> 이 연구에서는 이러한 모형을 수정하여 6가지 주거 공간의 배치로 표현된 단위평면을  $56 \times 56 \times 6$  행렬로 입력받아, 해당 평면이 계획된 시기를 5년 단위 10개 시기 중 하나로 판단하는 수정 VGG-GAP 모형을 주거 공간 배치 분석 모형으로 구축한다. 입력층과 출력층 뿐만 아니라 전체적인

---

<sup>67</sup>Zhou 등. (2015). Learning Deep Features for Discriminative Localization.

신경망의 규모도 이러한 작업의 난도에 맞추어 조정한다. 특히, 출력층 직전 계층에 위치한 GAP 계층의 규모를 512개 노드로 축소한다 (그림 5.1).

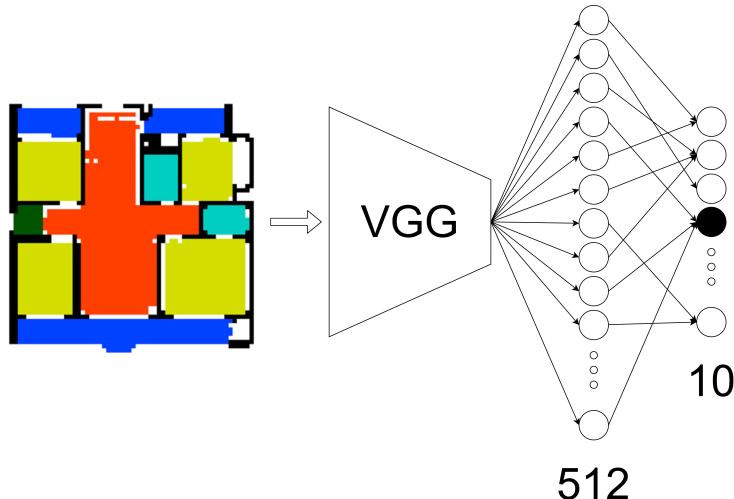


그림 5.1: 수정 VGG-GAP 모형 개념도

이 연구에서는 기계학습 및 딥 러닝 라이브러리인 텐서플로우 (Tensorflow)에서 제공하는 케라스 (Keras) API를 이용하여 수정 VGG-GAP 모형을 구현하였다. 최종적으로 구현된 모형의 구조는 그림 5.2와 같다.

다음으로, 이 모형이 한국 아파트 단위평면의 주거공간 배치를 보고 준공 시기를 예측할 수 있도록 4장에서 구축한 한국 아파트 단위평면 데이터셋을 학습시켰다. 학습 결과, 주거공간 배치 분석 모형은 한국 아파트 단위평면 50,252개 중 29,027개 평면에 대하여 5년 단위 시기를 정확히 예측하였다. 16,457개 평면에 대해서는 정확한 준공연도의 직전 시기 또는 직후 시기로 예측하였는데, 이를 포함하면 총 90.51%의 사례에 대하여 5년 이내의 오차로 준공 시기를 예측하였다 (그림 5.3). 이렇게 주거공간 배치 분석 모형이 시기별 단위평면을 잘 분류할 수 있다는 것은 각 시기의 단위평면에서만 나타나는 특징을 잘 학습하였다는 것을 의미한다.

이 장에서는 이와 같은 과정을 거쳐 한국 아파트에 대한 학습을 마친 모형을

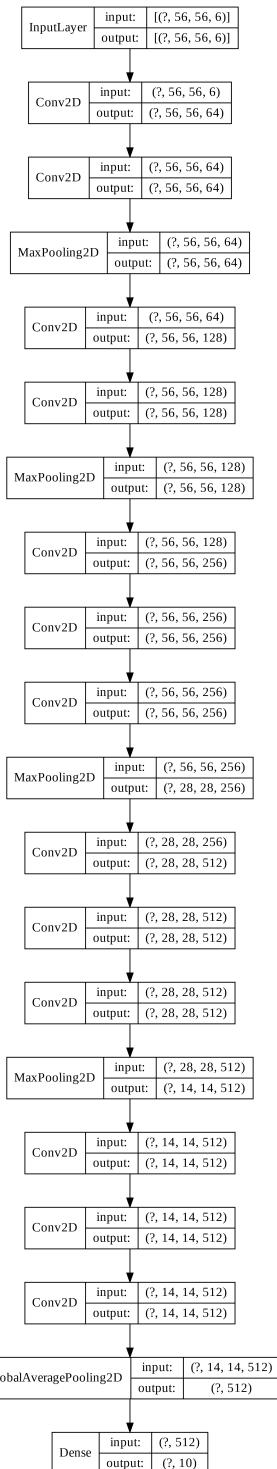


그림 5.2: 수정 VGG-GAP 모형 개요

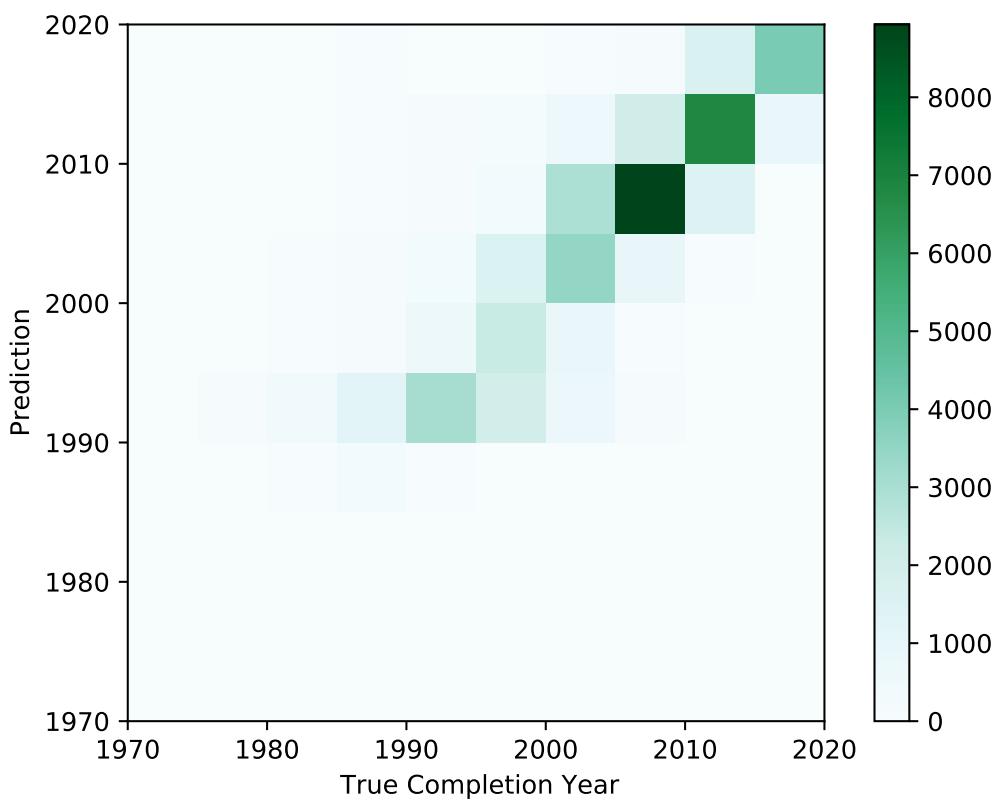


그림 5.3: 준공 시기 예측 결과

활용하여 한국 아파트에서 나타나는 단위평면 계획의 진화과정을 분석한다.

## 5.2 주거공간 배치유형 도출

한국 아파트 단위평면에서 나타나는 시기별 주거공간 배치를 학습한 딥 러닝 모형이 도출한 시기별 주거공간 배치 특성을 기반으로, 잠재표현에서 나타나는 평면 계획 특성과 이에 해당하는 단위평면 사례를 짹지어 유형화한다.

한국 아파트 단위평면을 시기별로 분류하는 작업을 학습한 수정 VGG-GAP 모형은 각 평면 사례에서 나타나는 계획 특성을 잠재표현 계층의 512개 노드를 활성화 (activation) 하는 강도로 판단한다. 즉, 이 모형은 주거공간 배치 행렬 데이터가 512가지 패턴 각각에 대하여 얼마나 정확하게 일치하는지 살핀 후, 그 결과를 기반으로 준공 시기를 예측하여 분류하는 것이다.

이렇게 학습된 잠재표현을 바탕으로, 이 모형은 91%의 사례에 대하여 작은 오차로 준공 시기를 예측할 수 있다. 따라서 수정 VGG-GAP 모형이 학습한 512가지 주거공간 배치 패턴으로 구성된 잠재표현은 시간에 따른 한국 아파트의 진화과정을 분석하기에 필요한 주거공간 배치 계획 특성을 담고 있다고 판단할 수 있다.

단위평면 유형화의 첫 단계로, 모든 단위평면에 대한 모든 잠재표현 노드의 활성화 값을 행렬의 형태로 표현한다. 잠재표현 활성화 행렬에서, 각 단위평면은 50,252개 행으로, 수정 VGG-GAP 모형에 각 단위평면을 입력시켰을 때 잠재표현 계층의 512개 노드가 활성화되는 값을 512개 열로 표현된다 (그림 5.4).

바이클러스터링 알고리즘인 spectral co-clustering은 이 행렬에 대하여 활성화 값이 높게 나타나는 단위평면 행과 잠재표현 노드 열의 짹을 양방향으로 찾아, 전체 행렬을 정해진 수의 바이클러스터로 나누게 된다.

바이클러스터링 분석에 앞서, 전체 단위평면을 몇 개의 유형으로 나누는지를

	0	1	2	3	4	5	6	7	8	9	...	502	503	504	505
0	0.142803	0.400475	0.134664	0.075232	0.161829	0.335772	0.034391	0.178221	0.228338	0.079080	...	0.218878	0.476947	0.042160	0.302460 0.2
1	0.153040	0.369468	0.343546	0.159964	0.177989	0.296884	0.026182	0.111398	0.232975	0.071310	...	0.136407	0.148069	0.051981	0.146548 0.2
2	0.105058	0.218708	0.427899	0.148000	0.038263	0.384796	0.042876	0.116903	0.126105	0.052608	...	0.233396	0.153381	0.012768	0.152117 0.1
3	0.126020	0.440626	0.223603	0.133866	0.113369	0.259057	0.200247	0.128830	0.304532	0.143829	...	0.191846	0.366552	0.021012	0.415277 0.2
4	0.158089	0.062936	0.228385	0.224262	0.195025	0.133689	0.054723	0.104017	0.008573	0.202741	...	0.084686	0.025523	0.125536	0.059776 0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
50247	0.024017	0.529761	0.392485	0.132199	0.021490	0.708420	0.004376	0.071726	0.014480	0.544205	...	0.359036	0.178483	0.105762	0.381127 0.0
50248	0.093362	0.533998	0.405788	0.156721	0.002338	0.579458	0.044787	0.176960	0.009811	0.499926	...	0.436967	0.315911	0.208208	0.438174 0.0
50249	0.014703	0.549547	0.378694	0.142741	0.018892	0.649889	0.018734	0.098782	0.029179	0.936829	...	0.336168	0.285051	0.175838	0.394663 0.0
50250	0.130718	0.577925	0.353213	0.089060	0.038444	0.597074	0.022201	0.076431	0.010245	0.433426	...	0.380023	0.363627	0.118584	0.550807 0.0
50251	0.074802	0.511034	0.240227	0.122885	0.022717	0.704477	0.056898	0.099791	0.059086	0.404906	...	0.409624	0.318817	0.200507	0.346482 0.0

50252 rows × 512 columns

그림 5.4: 잠재표현 활성화 행렬

의미하는 바이클러스터 수를 결정하여야 한다. 적절한 바이클러스터의 개수는 데이터에 따라 달라진다. 바이클러스터 개수가 너무 적으면 주거공간 배치의 차이를 충분히 표현하지 못하고, 너무 많으면 비슷한 단위평면이 별개의 유형으로 갈라지게 된다.

적절한 바이클러스터 수를 결정하기 위하여, 각 단위평면의 잠재표현 활성화 값을 기준으로 Mini-Batch K-Means 군집 분석을 수행하였다.<sup>68</sup> 군집의 수를 2개에서 30개까지 바꿔가면서, 군집의 수가 증가함에 따라 군집 분석 과정에서 생략된 정보의 크기를 의미하는 관성 (inertia, 군집 중심까지의 거리의 제곱 합) 지표의 값이 감소하는 속도를 측정한다. 이를 통하여, 바이클러스터를 하나 더 추가할 때 추가적으로 얻을 수 있는 정보의 양을 바탕으로 충분한 정보를 얻을 수 있는 적절한 바이클러스터의 개수를 결정한다.

그림 5.5에서, 군집 수에 따른 관성 지표 (파랑) 값이 군집 수가 하나 늘어날 때마다 감소하는 속도는 (빨강) 점차 감소하다가, 군집 수가 16개를 넘어선 이후로는 크게 감소하지 않는다. 즉, 16개보다 많은 바이클러스터에서 추가적으로 얻을 수 있는 정보가 크지 않음을 알 수 있다. 따라서 이 연구에서는 바이클러스터의 개수, 즉 주거공간 배치유형 수를 16개로 설정하였다.

<sup>68</sup>Sculley. (2010). Web-scale k-means clustering. In Proceedings of the 19th international conference on World wide web (pp 1177–1178).

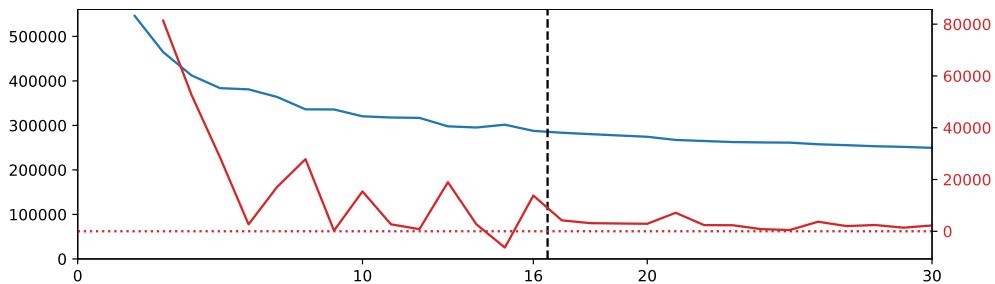


그림 5.5: 바이클러스터 수 결정

다음으로, 잠재표현 활성화 행렬에 대한 바이클러스터링을 통하여 한국 아파트 단위평면의 주거공간 배치유형을 도출하였다. 그림 5.6가의 잠재표현 활성화 행렬에 대하여 바이클러스터링 분석을 수행한 후, 바이클러스터별로 단위평면 행과 잠재표현 노드 열을 각각 정렬한 결과는 그림 5.6나와 같다. 이 때, 바이클러스터의 순서는 각 바이클러스터에 속한 단위평면 사례의 평균 준공연도를 기준으로 정렬하였다 (표 5.1).

표 5.1: 배치유형별 평균 준공연도

유형코드	평균 준공연도	유형코드	평균 준공연도
1	1989.4	9	2005.6
2	1992.4	10	2005.9
3	1993.3	11	2006.1
4	1996.5	12	2008.9
5	1997.4	13	2009.7
6	1999.6	14	2011.0
7	2000.0	15	2012.3
8	2004.4	16	2014.3

---

유형코드 평균 준공연도 유형코드 평균 준공연도

---

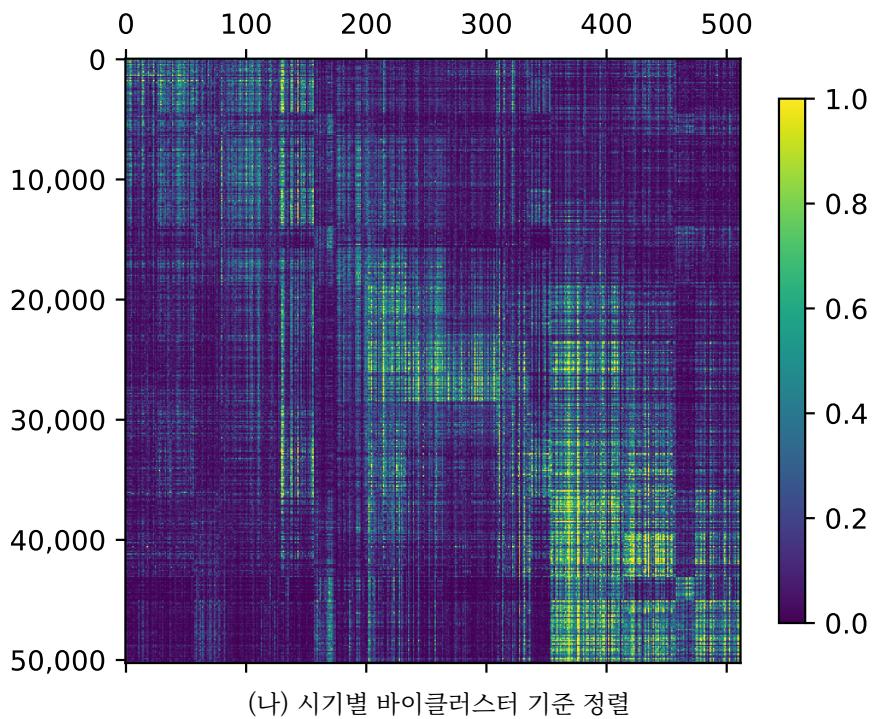
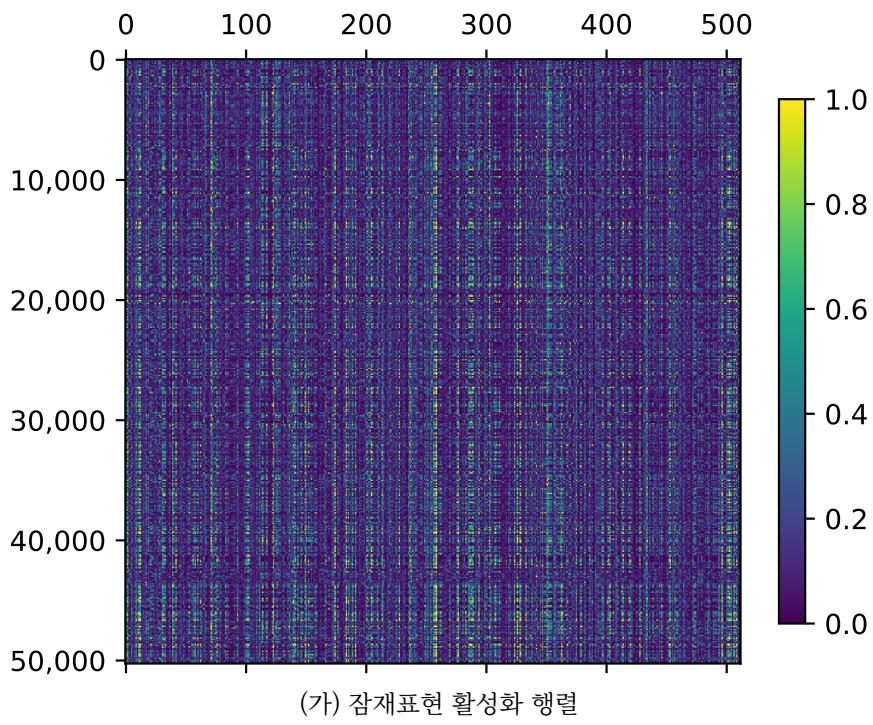


그림 5.6: 잠재표현 활성화 행렬 바이클러스터링

가상의 데이터에 대한 바이클러스터링 분석 결과를 보여주는 그림 3.4과 비교하면, 그림 5.6나의 바이클러스터링 결과는 단위평면과 잠재표현 노드의 활성화 사이에 1:1 관계가 정확하게 성립하지 않음을 알 수 있다. 현실에서는 하나의 단위평면에서 서로 독립적인 다양한 주거공간 배치 계획 특성이 나타날 수도 있고, 한 가지 계획 특성이 다양한 단위평면 계획 유형에 걸쳐 적용될 수도 있다. 따라서 여기서 도출된 주거공간 배치유형은 특정한 계획 유형과 그 특성이 가장 명확하게 나타나는 단위평면이 짹지어진 관계로 해석하여야 한다. 또한, 이러한 유형들이 서로 연관되는 패턴을 분석하여 현실에서 여러 계획 특성이 모여 하나의 상위 유형을 이루는 연관관계를 파악할 필요가 있다.

그림 5.7은 각 유형에 해당하는 단위평면에서 유형별 모형에 해당하는 잠재표현 노드의 평균 활성화 강도를 나타낸 것이다. 각 행은 해당 유형에 속한 단위평면, 각 열은 해당 유형에 속한 잠재표현 노드를 나타낸다. 단위평면과 잠재표현 노드 사이에 1:1 관계가 나타난다면, 행과 열의 번호가 일치하는 대각선 성분에서만 높은 활성화 (밝은 노랑)를 보이고, 나머지 영역은 낮은 활성화 (어두운 파랑) 정도가 나타나게 된다. 바이클러스터링의 결과로, 단위평면과 잠재표현이 서로 동일한 유형에 해당하는 대각선 성분이 다른 행이나 열에서보다 높게 나타나는 것은 당연하다. 그러나, 서로 다른 유형의 단위평면과 잠재표현 노드 사이의 관계를 의미하는 행과 열의 번호가 일치하지 않는 영역에서도 높은 활성화를 보이는 경우가 많이 나타났다. 특히 일부 유형에서는 다양한 행이나 열에서 높은 활성화가 나타나는 것을 알 수 있다. 예를 들어, 13번 열의 잠재표현 노드는 8번-16번 행의 단위평면 군집에 대하여 높은 활성화 정도를 보였다. 또한, 그 역의 관계에 해당하는 13번 행과 8번-16번 열의 영역에서는 그러한 활성화가 나타나지 않았으므로, 이러한 결과를 서로 다른 유형 사이에서 여러 유형에 걸친 1:1 관계가 나타났다고 해석할 수도 없다. 또, 그렇게 높은 활성화를 보이는 단위평면의 유형과 잠재표현의 유형은 멀리 떨어진 유형 (예: 1번 유형과 14번 유형)에서보다는 인접한 유형 (예: 1번과 2번, 13번과

14번)인 경우가 많다는 것도 확인할 수 있다.

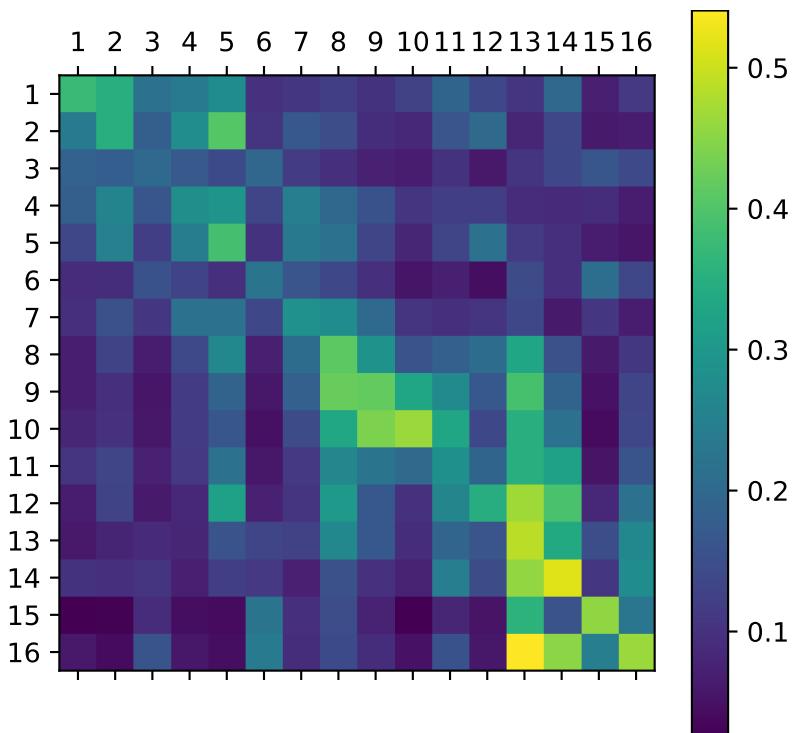


그림 5.7: 유형별 모형의 평균 활성화 강도

각 주거공간 배치유형이 평균 준공연도로 정렬되었다는 점을 고려하면, 인접한 유형 사이에서 높은 활성화를 보인다는 것은 유사한 시기에 계획된 다양한 단위평면 사이에 여러 계획 특성이 같이 나타나는 경우가 많다는 의미이다. 예를 들어, 13번 유형의 잠재표현 노드는 8번부터 16번까지 다양한 유형의 단위평면에 걸쳐 일반화된 주거공간 배치 패턴을 인식하는 유형이다. 다른 유형에서 이렇게 극단적인 예는 드물지만, 반대로 단 한 유형의 단위평면에만 활성화되는 잠재표현 노드도 거의 없다.

이렇게 여러 유형 사이에서 나타나는 연관성을 분석하기 위하여 각 유형의 잠재표현 활성화 강도에 대한 상관분석을 수행하였다. 모든 단위평면에 대하여, 각 유형별 잠재표현 노드에 대한 활성화 값의 합계를 해당 유형의 활성화 강도

로 정의하고, 서로 다른 유형이 모든 단위평면에서 활성화되는 강도 사이에서 나타나는 상관계수를 그림 5.8과 같이 나타내었다. 두 주거공간 배치유형의 상관계수가 높을수록 두 유형이 인식하는 주거공간 배치 패턴이 동일한 단위 평면에서 나타난다는 의미이다. 반대로 상관계수가 0보다 작은 경우, 한 유형의 주거공간 배치가 나타나는 단위평면에서는 다른 유형의 주거공간 배치가 나타나지 않을 가능성이 더 커진다는 것을 의미한다.

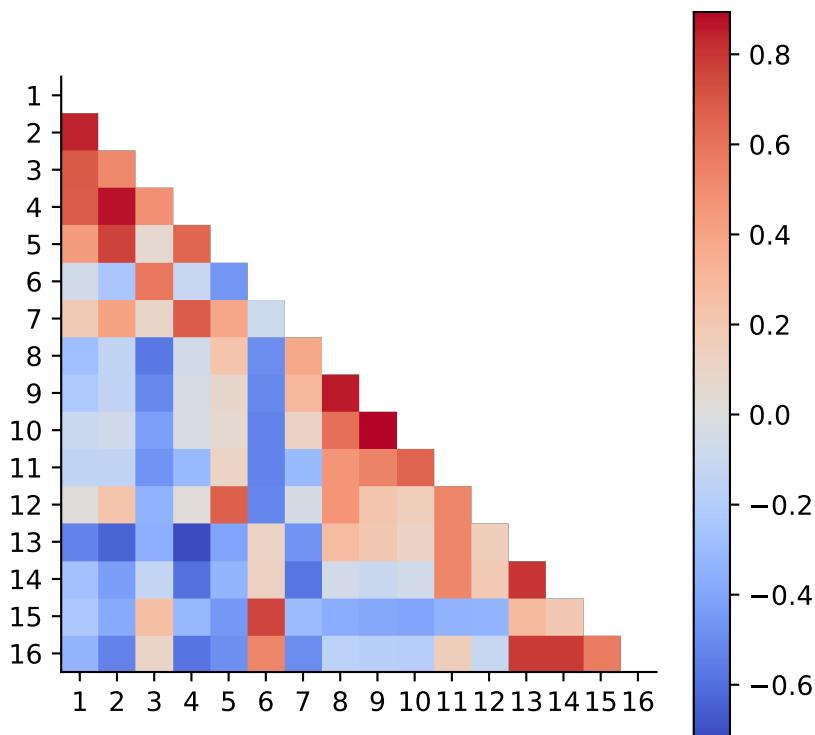


그림 5.8: 유형별 모형의 활성화 강도에 대한 상관분석

상관분석 결과에서, 서로 인접한 배치유형 사이에는 양의 상관관계가 나타나는 경우가 많음을 확인할 수 있다. 따라서 한국 아파트 단위평면과 시기 구분 모형의 잠재표현이 보이는 관계는 1:1 관계로 해석하기 어려움을 확인할 수 있다. 그 안에서도 1번에서 5번까지, 8번에서 11번까지, 13번에서 16번까지 등 3개 집단에서 특히 높은 상관관계가 군집을 이루어 나타난다. 이러한 군집은

유사한 시기의 단위평면에 대하여 여러 계획 특성이 동시에 나타나는 상위 평면 계획 유형의 존재를 시사한다.

한편, 5번과 12번, 6번과 15-16번은 평균 준공연도가 10년 이상 떨어져있는데도 양의 상관관계를 보인다. 이러한 관계는 특정 시기에 한정되지 않고 다양한 시기의 단위평면에서 나타나는 주거공간 배치 특성을 의미할 수 있다. 그러나 그림 5.7에서 각 유형의 단위평면과 잠재표현 노드의 관계를 검토하면, 인접한 유형의 활성화에 비하면 그 강도가 약한 것을 확인할 수 있다. 따라서 해당 유형 사이의 관계에 대한 해석은 5.3절에서 BAM 분석을 통하여 실제 주거공간 배치 특성을 확인한 이후에 진행한다.

이와 반대로, 시기적으로 많이 떨어진 배치유형 사이에는 음의 상관관계가 나타난다. 3번, 6번, 15번 유형의 경우는 그 정도가 특히 두드러져, 각 유형이 속한 군집을 제외하면 대부분의 유형과의 관계에서 음의 상관관계가 나타난다. 이 세 유형은 다른 단위평면들과 유사한 주거공간 배치를 공유하지 않는 비교적 독립적인 단위평면 계획에 대한 유형임을 파악할 수 있다.

### 5.3 주거공간 배치유형별 계획 특성 분석

5.2절에서 도출된 주거공간 배치유형의 성격을 분석하고, BAM 시각화를 통하여 각 유형의 계획 특성을 해석하였다.

먼저 각 배치유형에 속한 단위평면 사례의 준공 시기는 그림 5.9과 같았다. 한국 아파트 단위평면을 시기별로 분류할 수 있도록 딥 러닝 모형을 학습시켰으므로, 이러한 작업을 수행할 수 있도록 학습한 딥 러닝 모형의 잠재표현 계층은 시기별로 다르게 나타나는 주거공간 배치 특성을 재현하게 된다. 따라서 이러한 딥 러닝 모형의 잠재표현에서 도출한 배치유형에서도 각 시기별 단위평면 사례가 서로 다른 배치유형으로 잘 묶여 나타났다.

1번부터 16번 유형까지 각 시기별 단위평면 사례의 비중이 연속적으로 변화하지만, 크게 1980년대 평면이 포함되는 유형이 3개 (1번-3번), 그 다음 1990년대 평면이 포함되는 유형이 4개 (4번-7번), 거의 대부분의 사례가 2000년대 평면인 유형이 4개 (8번-11번), 2000년대와 2010년대에 걸친 유형이 3개 (12번-14번), 그리고 마지막으로 2010년대 사례가 대부분인 유형이 2개 (15번-16번)로 나눌 수 있다.

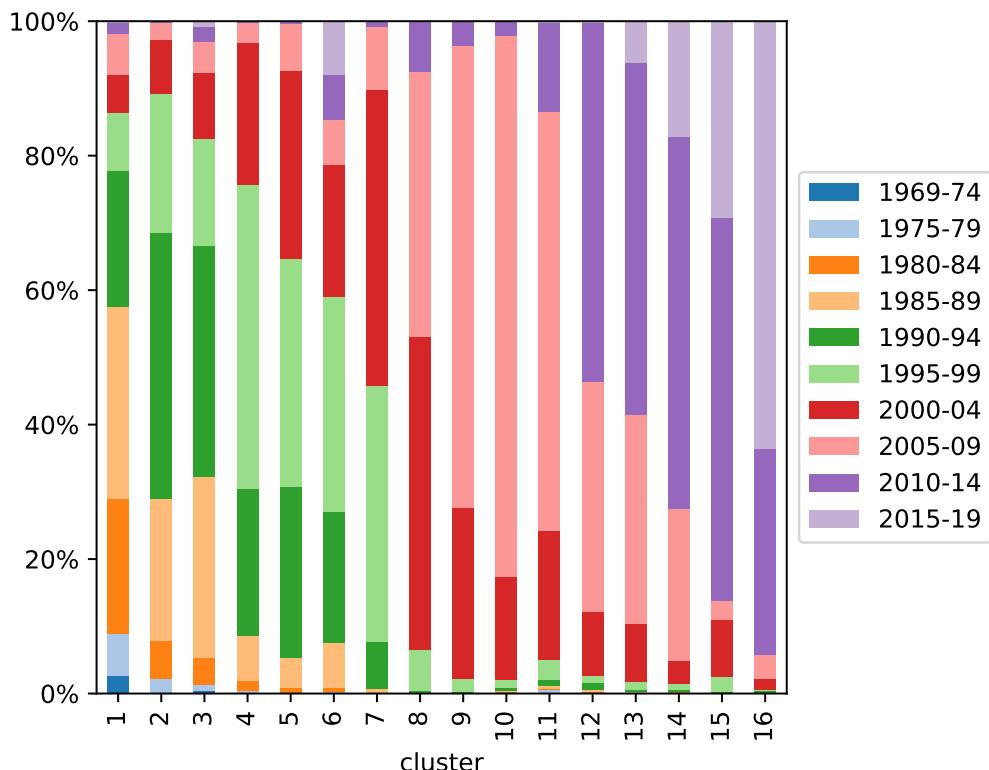


그림 5.9: 배치유형별 단위평면 시기 분포

단위평면 사례의 시도별 분포는 배치유형에 따라 크게 차이가 나지는 않는다 (그림 5.10). 서울 및 경기의 수도권 비중이 시간의 흐름에 따라 점차 감소하지만, 가장 최근에 해당하는 16번 유형에서도 여전히 절반 가량을 차지하였다. 이러한 결과는 지역에 따라 단위평면 계획이 다르게 나타나지는 않는다는 의미로 해석할 수 있다.

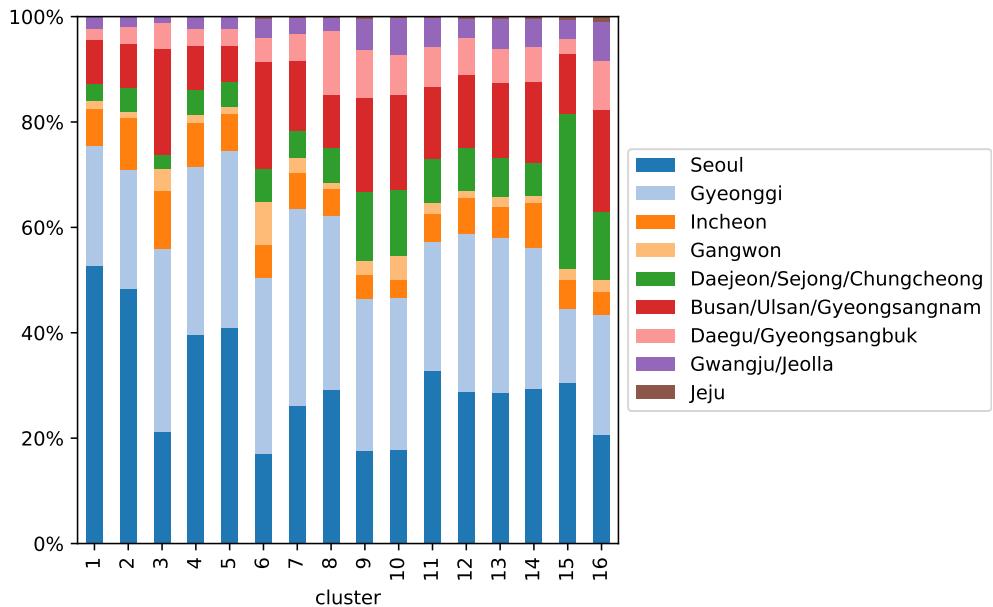


그림 5.10: 배치유형별 단위평면 지역 분포

국민주택 규모와 원룸형 도시형생활주택 기준을 적용하여 원룸형 ( $50\text{ m}^2$  이하), 소형 ( $60\text{ m}^2$  이하), 중형 ( $85\text{ m}^2$  이하), 대형 ( $85\text{ m}^2$  초과) 으로 나누어 분석한 규모별 분포에서는, 15번 유형의 거의 대부분이 원룸형 규모에 해당함을 알 수 있다. 시기적으로도 원룸형 도시형생활주택의 등장 시기 (2009년)과 일치하므로, 도시형생활주택이 별도 유형으로 분류되었음을 파악할 수 있다. 그 외 소형 이하의 비중이 절반 이상을 차지하는 유형도 일부 있지만 (3번, 6번, 7번), 대부분의 유형에서는 국민주택 규모와 그 이상의 대형 평면이 분류되지 않고 고르게 나타났다.

침실 수와 화장실 수에서도 원룸형 도시형생활주택 유형을 제외하면 침실 3개 화장실 2개 평면이 주류로 나타났다. 1번부터 7번 유형까지, 즉 1990년대까지의 단위평면이 포함되는 유형에서는 화장실 1개 사례의 비중이 높은 편이나, 2000년대 이후로는 거의 찾아보기 어렵다.

다음으로, 배치유형별 단위평면 사례에 대한 BAM 시각화를 통하여 각 유형의

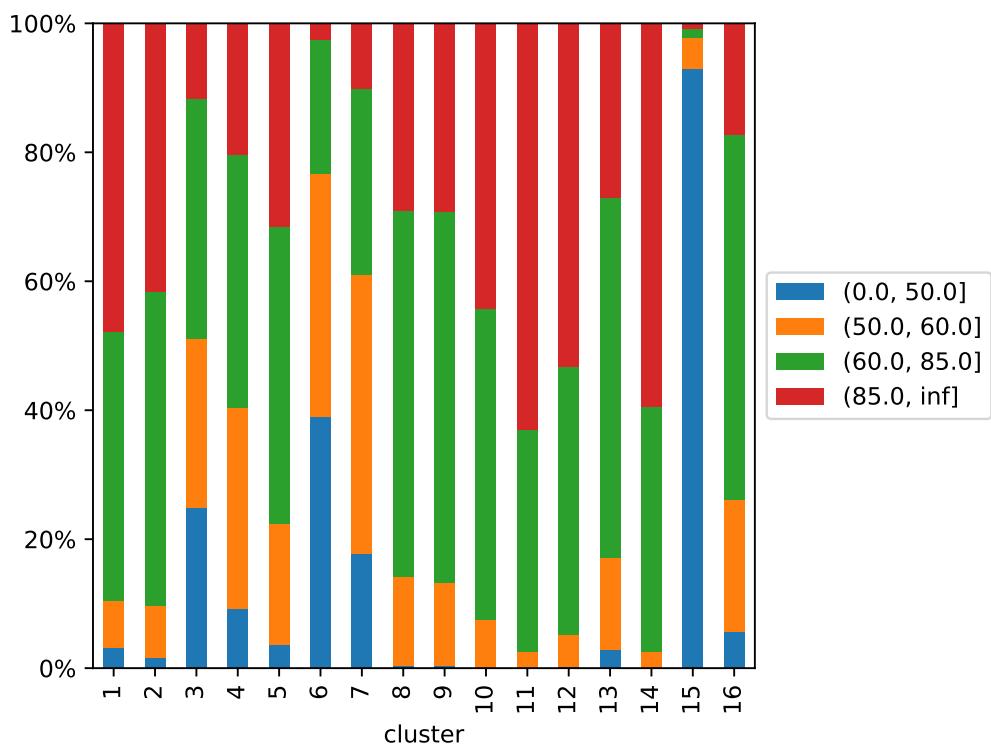


그림 5.11: 배치유형별 단위평면 규모 분포

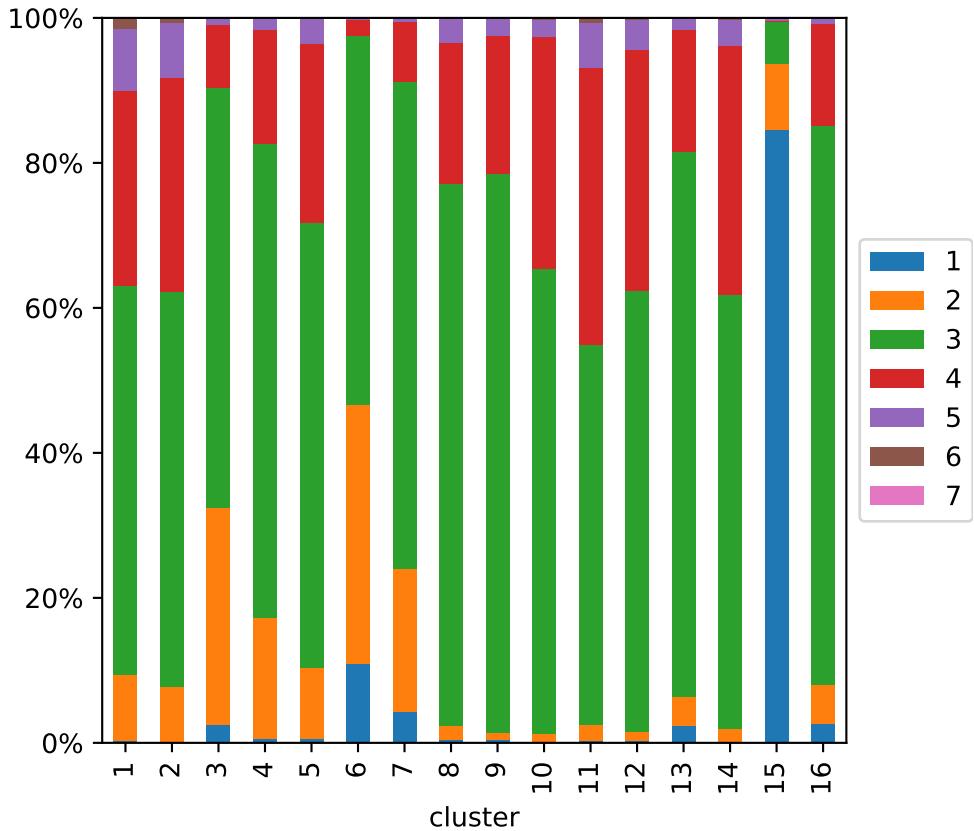


그림 5.12: 배치유형별 단위평면 침실 수 분포

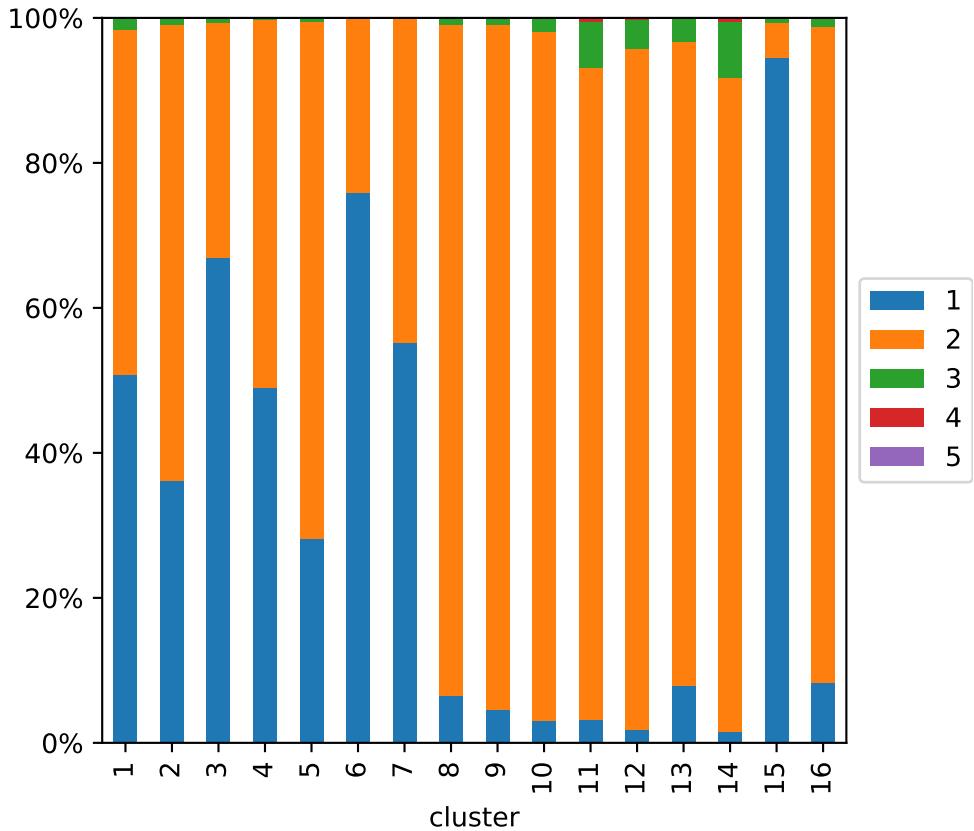


그림 5.13: 배치유형별 단위평면 화장실 수 분포

주거공간 배치 특성을 분석하였다. 또한, 단위평면의 규모에 따라 배치유형이 인식하는 특성이 어떻게 달라지는지를 함께 살펴보았다. 해당 시기의 여러 규모의 단위평면에서 딥 러닝 모형이 발견한 주거공간 배치 특성을 해석하였다. 주거공간 배치유형과 단위평면 규모에 따른 단위평면 사례에 대한 각 유형의 BAM 시각화 결과는 그림 5.14과 같다.

각 배치유형의 이름은 해당 유형의 단위평면 사례가 나타나는 시기를 기준으로 하고, 같은 시기 평면에 대한 유형들은 일련번호를 통하여 구분하였다. 5.2절에서 살펴보았듯이, 같은 시기 평면에 대한 여러 배치유형은 평면과 유형 사이에 1:1 관계가 강하게 나타나지 않기 때문에, 각각의 유형을 명확하게 구분할 수 있는 이름을 정성적으로 붙이는 것이 어려웠기 때문이다.

8090-1 유형은 (그림 5.15) 1980년대부터 1990년대까지 걸쳐 나타난 3개 배치유형 중 평균 준공연도가 가장 앞서는 유형이다. 이 유형의 단위평면에서는 다른 유형보다 판상형 주동의 복도형 단위평면이 집중되어 나타나는 것이 이 유형의 가장 큰 특징이다.

복도형 단위평면과 계단실형 단위평면은 현관의 위치를 통하여 정확하게 구분 할 수 있지만, 8090-1 유형을 정의하는 주거공간 배치 계획 특성은 그보다는 간접적으로 복도형 단위평면을 인식한다. 4.3.1절에서 정의한 방향 정규화 기준에 따라 처리된 단위평면에서 계단실형 단위평면의 현관은 왼쪽 중앙에 위치하게 된다. 그러나 복도형 단위평면에서는 현관이 위쪽 후면에 위치하고, 그 자리에는 다른 주거 공간이 표현된다. 8090-1 유형은 현관 대신 그 위치에 계획된 화장실을 보고 복도형 단위평면을 다른 계단실형 평면과 구분한다.

그 외에도, 거실 전면에만 발코니가 계획되어 입면에서 돌출되거나, 전면 발코니에 배관 공간이 계획되는 등, 1980~1990년대 복도형 단위평면에서 나타나는 여러 특징을 함께 인식한다.

8090-2 유형은 (그림 5.16) 계단실형 단위평면의 전형적인 배치를 인식한다.

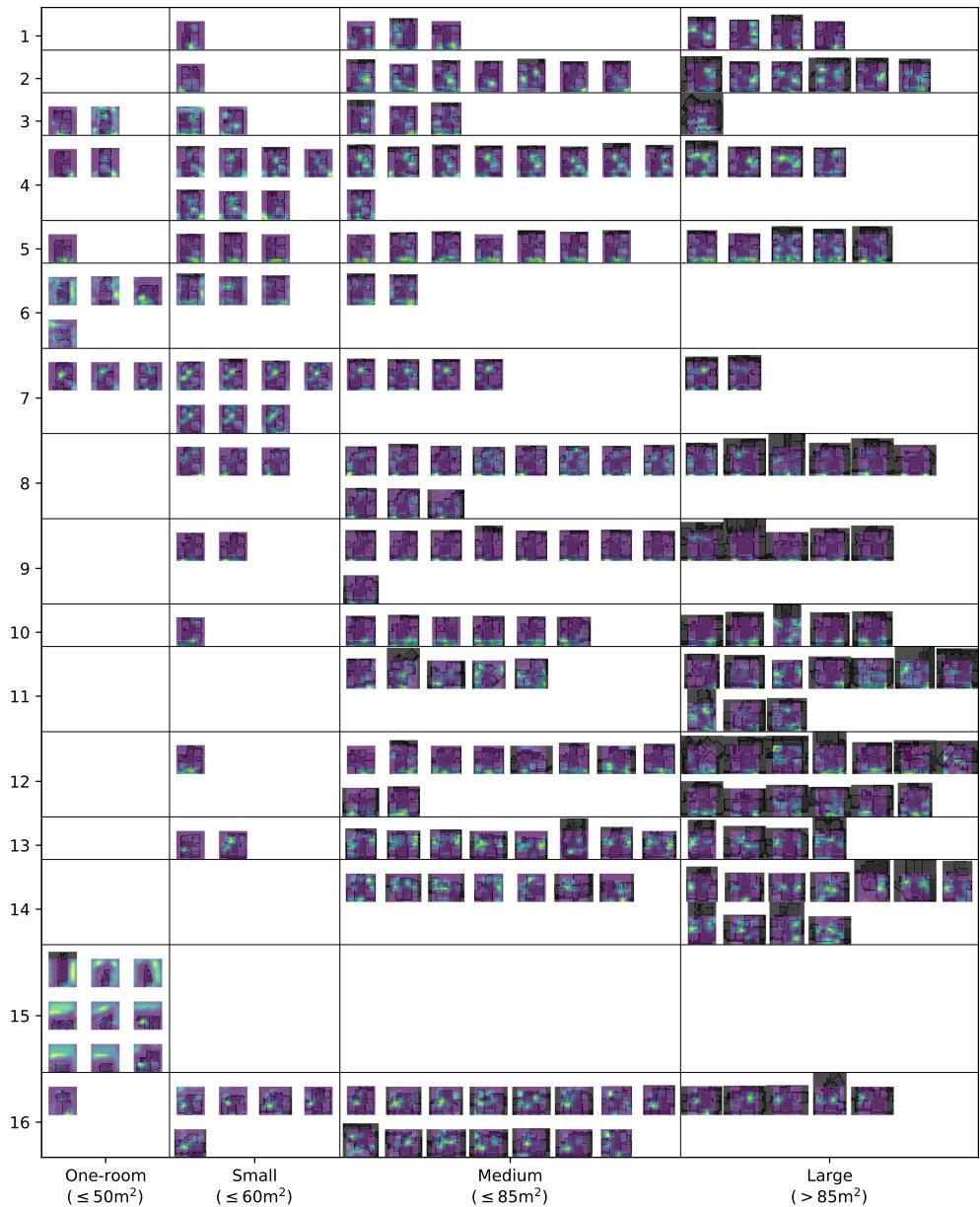
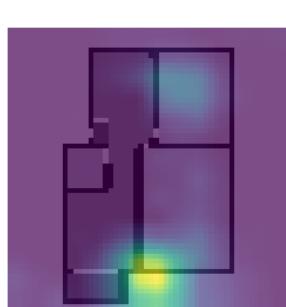
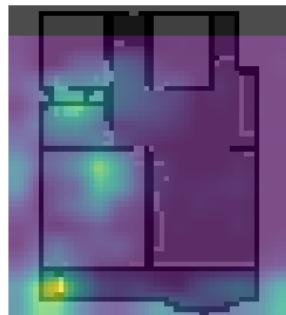


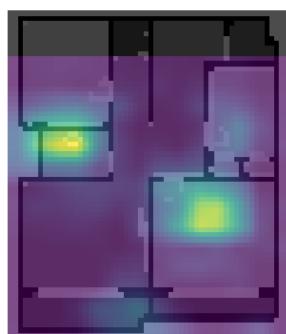
그림 5.14: 배치유형별, 규모별 단위평면 BAM 시각화



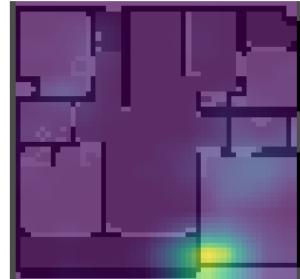
(가) 거실 전면 돌출 밸코니



(나) 복도형 (화장실), 전면 밸코니 설비공간



(다) 복도형 (화장실), 주 침실



(라) 거실 전면 돌출 밸코니

그림 5.15: 8090-1 유형의 주거공간 배치 계획 특성

베이 전체를 차지하는 계단실에서 이어지는 현관이 단위평면의 측면에 배치되면서, 주 침실은 현관에서 먼 쪽 전면에 배치된다. 4LDK 평면에서는 거실과 주방/식당 사이에 두 번째 화장실을 계획하여 전후면 전체를 외기에 개방한다. 그러나 8090-1 유형과 마찬가지로 전면 발코니에 배관 공간이 계획된다.

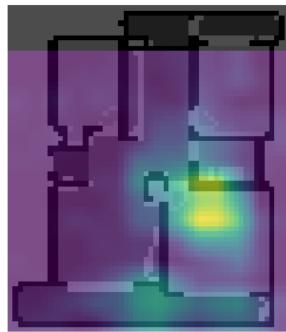
8090-3 유형은 (그림 5.17) 소형 단위평면 중 가장 앞선 유형이다. 5.2절의 상관분석에서 다른 대부분의 유형과 음의 상관관계를 보였던 8090-3, 9000-3, 10-1 유형은 모두 소형 단위평면이 집중되어 나타난다. 딥 러닝 모형에 주어진 과제는 시기별로 단위평면을 분류하는 것이었지만, 그 과정에서 평면 규모별로도 분류하는 법을 학습한 것이다.

8090-3 유형은 분석 영역보다 작은 단위평면 외부의 여백을 해당 유형의 특징으로 인식한다. 이러한 경우 작다는 사실 자체가 평면 내부의 주거공간 배치보다 더 중요하게 취급되는 문제가 있다. 그렇기 때문에 가장 많은 사례가 존재하는 국민주택 규모에 맞추어 분석 영역의 크기를 설정하였지만, 그렇게 해도 그보다 작은 소형 단위평면의 주거공간 배치는 많은 부분 무시된다.

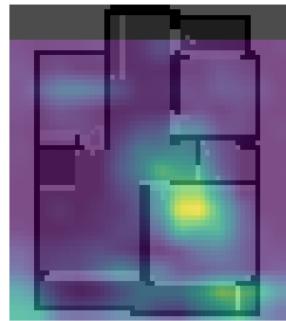
이 유형에서 나타나는 또 하나의 특징은 전면 발코니의 개방면이 베이를 기준으로 분절되어 있다는 것이다. 다른 대부분의 단위평면에서처럼 전면 전체가 하나의 개방면으로 이루어지지 않고 베이를 기준으로 외벽이 표현된 것을 인식하고, 그 사이의 발코니 경계를 이 유형의 계획 특성으로 정의하고 있다.

9000-1 유형은 (그림 5.18) 1990년대부터 2000년대에 걸쳐 나타나는 4개 유형 중 첫 유형이다. 이 유형은 1990년대의 전형적인 3LDK 평면을 인식한다. 이 계단실형 단위평면은 전면 2베이 중 현관에서 먼 쪽에 주 침실이 계획되고, 후면 3베이의 중앙에는 화장실 등으로 가로막히지 않은 주방/식당이 배치된다. 또한, 전면 폭 전체에 걸쳐 발코니가 계획된다.

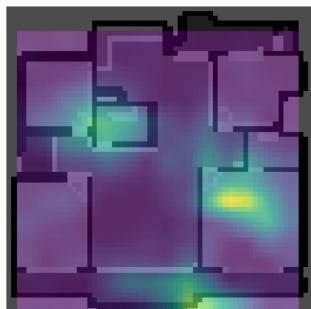
9000-2 유형은 (그림 5.19) 전면 폭 전체에 걸친 발코니를 인식한다. 현실에서는 전면 발코니가 계획된 단위평면과 다른 여러 유형의 단위평면이 따로 존



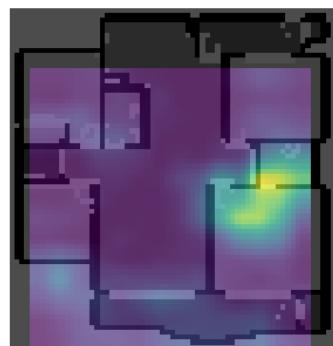
(가) 현관 면 쪽 주 침실



(나) 주 침실, 전면 발코니 설비공간

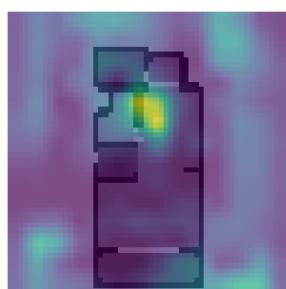


(다) 주 침실, 중앙 화장실, 거실 발코니

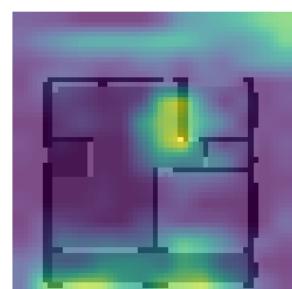


(라) 현관 면 쪽 주 침실과 화장실

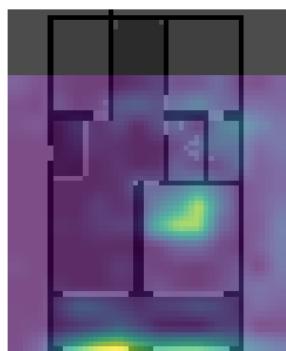
그림 5.16: 8090-2 유형의 주거공간 배치 계획 특성



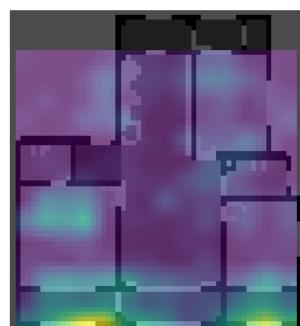
(가) 소형, 중앙 DK



(나) 소형, 중앙 DK, 전면 발코니 베이

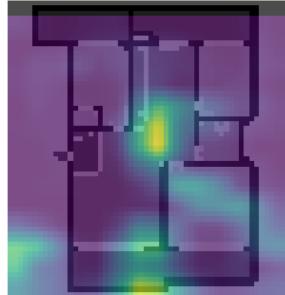


(다) 전면 발코니 베이, 주 침실

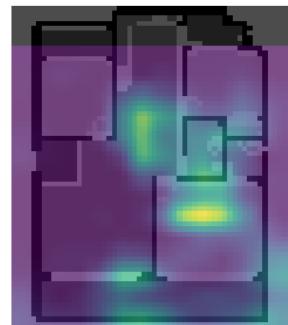


(라) 전면 발코니 베이

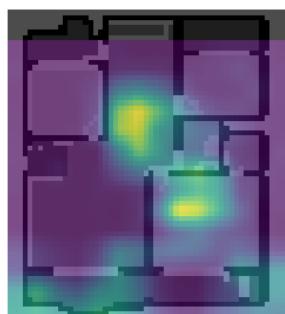
그림 5.17: 8090-3 유형의 주거공간 배치 계획 특성



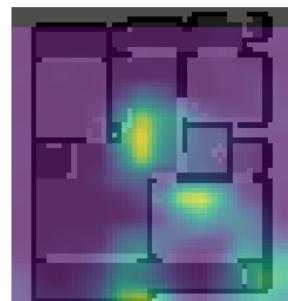
(가) 후면 중앙 DK, 전면 발코니



(나) 전면 주 침실, 후면 중앙 DK



(다) 전면 주 침실, 후면 중앙 DK



(라) 주 침실, 중앙 DK, 전면 발코니

그림 5.18: 9000-1 유형의 주거공간 배치 계획 특성

재하는 것이 아니지만, 이 연구에서는 전면 전체 발코니를 인식하는 잠재표현 노드를 가장 잘 설명할 수 있는 단위평면이 이 유형으로 분류되었다.

9000-3 유형은 (그림 5.20) 소형 단위평면이 인식된 두 번째 유형이다.  $50\text{ m}^2$  이하 원룸형 규모와  $60\text{ m}^2$  이하 소형을 아울러 좁은 전면 폭을 인식한다. 그 외에 전면 발코니도 이 유형의 특징으로 나타나지만, 여백의 규모가 더 큰 원룸형 규모에서는 전면 발코니가 있더라도 잘 인식되지 않는다.

9000-4 유형은 (그림 5.21) 9000-1 유형과 마찬가지로 전면 2베이 후면 3베이 3LDK 평면을 인식한다. 후면 중앙 주방/식당의 배치를 인식하는 것은 유사 하지만, 주 침실이 아닌 측면 현관과 거실의 배치로 전면 배치를 인식한다. 또한, 소형 이하 비중이 60%에 달하여, 중형 이상이 60% 였던 9000-1 유형과 차이를 보인다.

00-1 유형과 00-2 유형은 거의 대부분의 사례가 2000년대에 집중된 4개 유형 중 두 유형이다. 이 두 유형은 단위평면 전체에서 단 한 지점만을 인식하는 유형이다. 전면 전체에 발코니가 계획되지만 거실 발코니의 폭이 더 넓어, 현관 쪽 침실 전면 발코니보다 돌출되는 지점을 인식한다 (그림 5.22, 5.23).

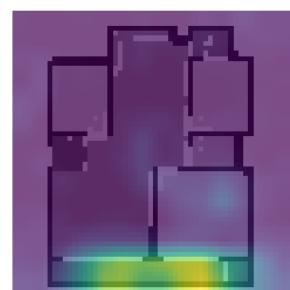
한편 00-3 유형은 (그림 5.24) 전면 발코니의 깊이 방향 폭이 넓게 나타나는 단위평면을 인식한다. 이렇게 2000년대에만 해당하는 4개 유형 중 3개가 발코니에만 집중하고 있다는 점은 해당 시기의 발코니 법규 변화와 함께 분석할 필요가 있다.

00-4 유형은 (그림 5.25) 단위평면의 전면 폭이 넓게 나타나면서 침실과 화장실 등 양쪽의 주거 공간이 분석 영역이 넘치게 양 끝까지 표현되는 것을 인식한다. 00-4 유형에 해당하는 단위평면은  $85\text{ m}^2$  초과 대형 평면이 60%에 달하므로, 넓은 전면 폭을 통하여 대형 평면을 인식한다고 볼 수 있다.

0010-1 유형은 (그림 5.26) 2000년대부터 2010년대까지 나타난 유형이다. 이 유형은 전면 폭 전체에 걸친 발코니가 분석 영역을 가득 채우는 것을 인식한다.



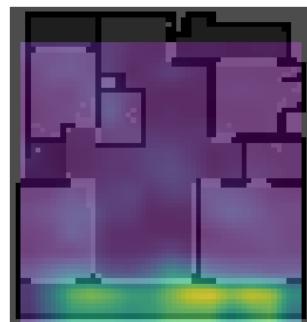
(가) 전면 폭 전체 밸코니



(나) 전면 폭 전체 밸코니

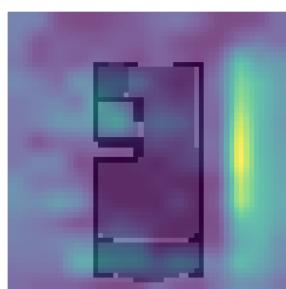


(다) 전면 폭 전체 밸코니

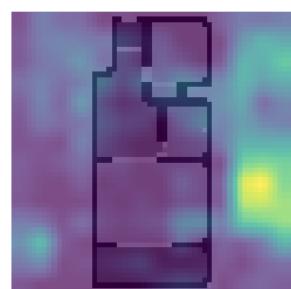


(라) 전면 폭 전체 밸코니

그림 5.19: 9000-2 유형의 주거공간 배치 계획 특성



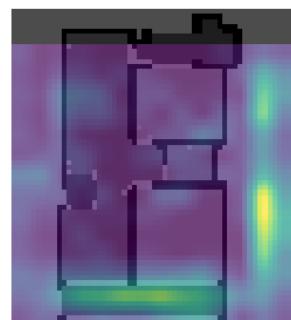
(가) 원룸형 (좁은 전면 폭)



(나) 원룸형 (좁은 전면 폭)

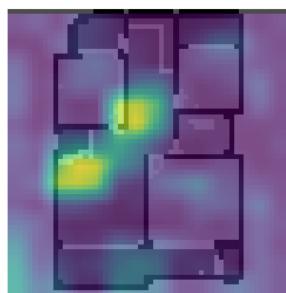


(다) 작은 침실, 전면 밸코니

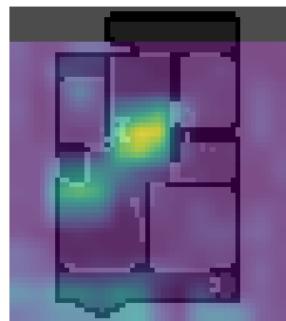


(라) 소형 (좁은 전면 폭), 전면 밸코니

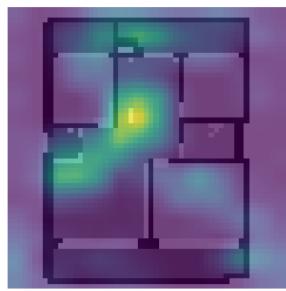
그림 5.20: 9000-3 유형의 주거공간 배치 계획 특성



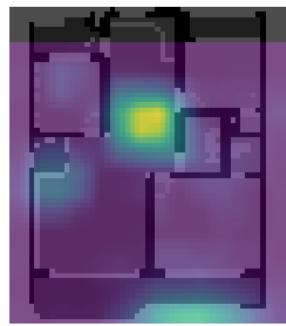
(가) 후면 중앙 DK, 소형 (현관)



(나) 후면 중앙 DK, 소형 (현관)

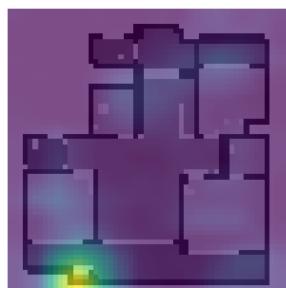


(다) 후면 중앙 DK, 소형 (현관)

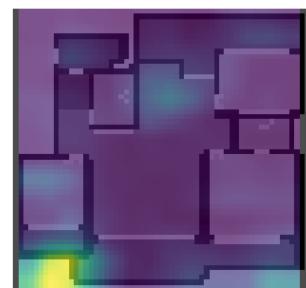


(라) 후면 중앙 DK, (중형)

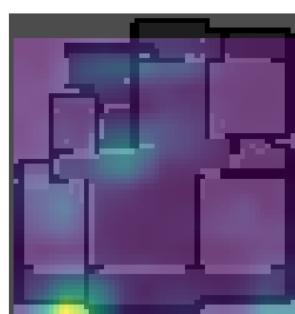
그림 5.21: 9000-4 유형의 주거공간 배치 계획 특성



(가) 발코니 폭 차별화



(나) 발코니 폭 차별화



(다) 발코니 폭 차별화

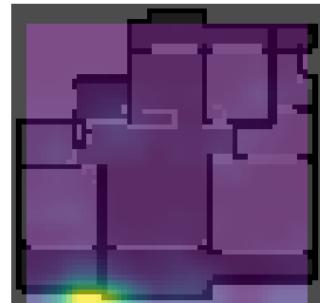


(라) 발코니 폭 차별화

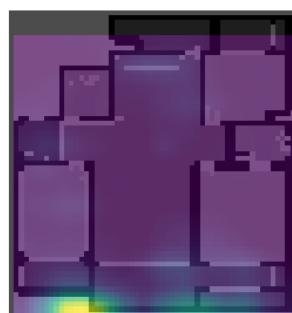
그림 5.22: 00-1 유형의 주거공간 배치 계획 특성



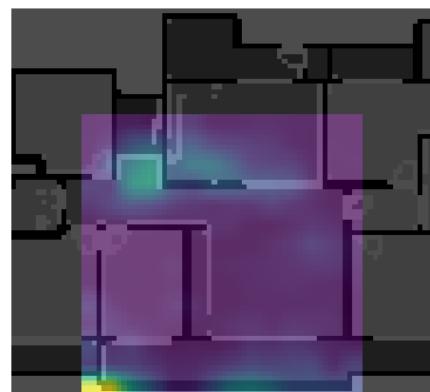
(가) 발코니 폭 차별화



(나) 발코니 폭 차별화

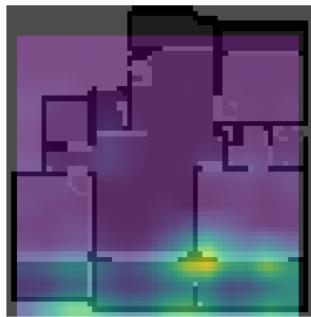


(다) 발코니 폭 차별화

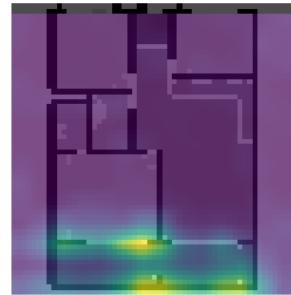


(라) 발코니 폭 차별화

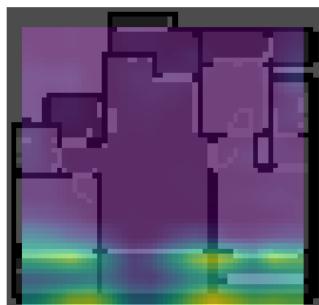
그림 5.23: 00-2 유형의 주거공간 배치 계획 특성



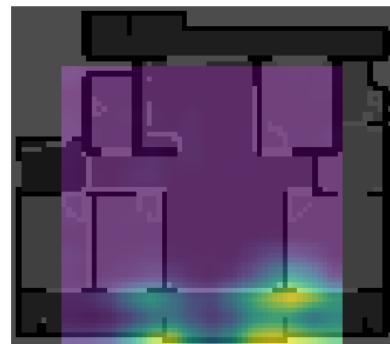
(가) 넓은 발코니 폭



(나) 넓은 발코니 폭

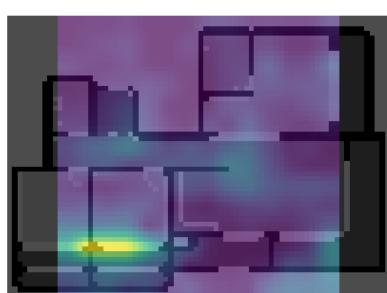


(다) 넓은 발코니 폭

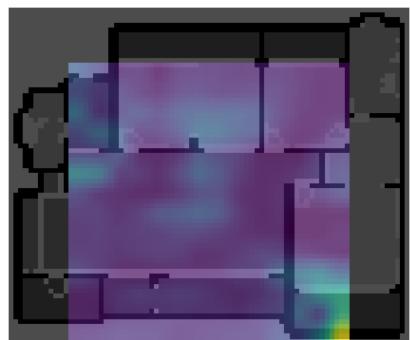


(라) 넓은 발코니 폭

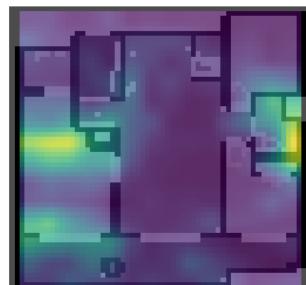
그림 5.24: 00-3 유형의 주거공간 배치 계획 특성



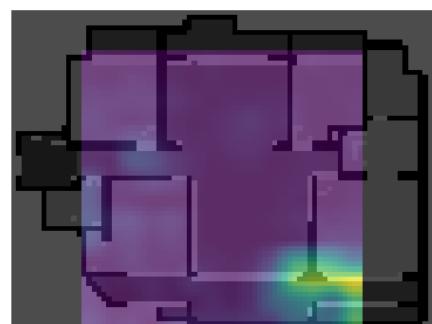
(가) 넓은 전면 폭 (침실 위치)



(나) 넓은 전면 폭 (침실 위치)



(다) 넓은 전면 폭 (침실, 화장실)



(라) 넓은 전면 폭 (침실 위치)

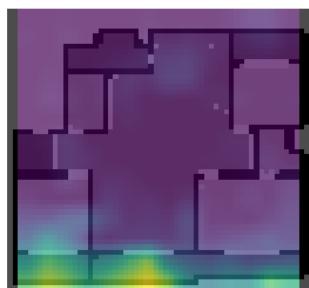
그림 5.25: 00-4 유형의 주거공간 배치 계획 특성

0010-2 유형도 넓은 전면 폭을 인식하는 것은 마찬가지이지만, 그 결과로 단위 평면에 생겨난 복도를 함께 인식한다. (그림 5.27). 0010-3 유형에는 같은 시기의 다른 유형들보다 탑상형 평면이 집중되어 나타나고, 그러한 단위평면에서 나타나는 주거공간 배치를 인식하지만, 그 경우에도 복도는 주거공간 배치의 특성을 정의하는 특징이 된다 (그림 5.28).

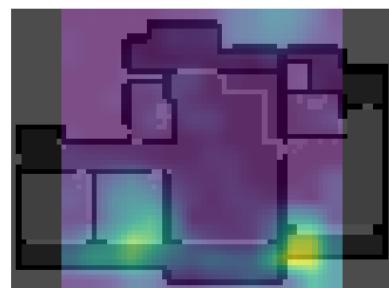
10-1 유형은 2010년대에 들어 등장한 두 유형 중 하나이다. 10-1 유형은 절대 다수가 원룸형 규모로, 다른 두 소형 유형보다도 더 작은 단위평면의 여백과 작은 침실을 인식하여 이 유형을 구분한다 (그림 5.29). 단위평면이 더 작다는 것은 그만큼 여백의 중요도가 높아진다는 의미로, 규모를 제외한 다른 특징은 발견하기 어렵다.

10-2 유형은 2010년대 들어 등장한 다른 한 가지 유형이다. 이 시기의 단위 평면은 넓은 전면 폭과 얕은 깊이를 보인다. 단위평면 내부에서 전면 폭과 함께 LDK의 폭은 넓어지지만 복도도 함께 길어지며, 얕은 깊이 때문에 침실의 규모는 매우 작아진다.

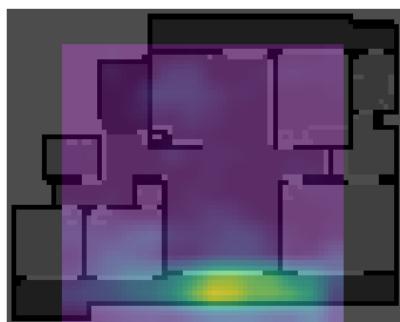
이상과 같이 BAM 시각화를 통하여 분석한 배치유형의 계획 특성을 정리하면 표 5.2과 같다.



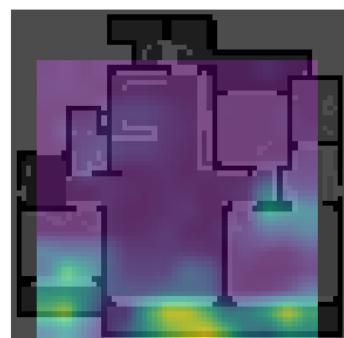
(가) 전면 발코니



(나) 넓은 전면 폭 (침실 위치)

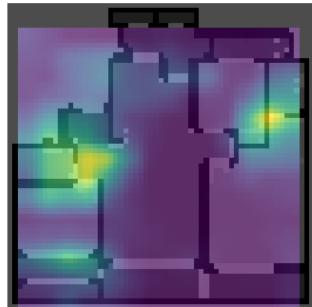


(다) 전면 발코니

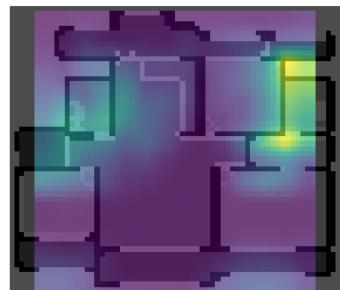


(라) 전면 발코니, 주 침실

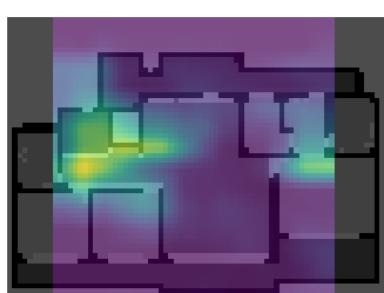
그림 5.26: 0010-1 유형의 주거공간 배치 계획 특성



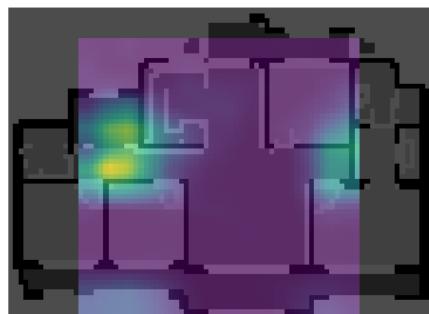
(가) 현관, 주 침실 화장실



(나) 후면 주 침실 화장실

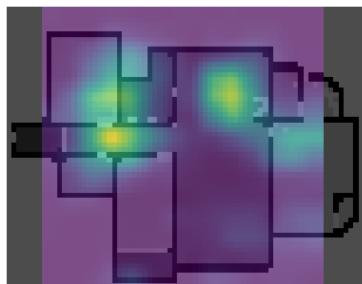


(다) 현관 쪽 복도, 주 침실

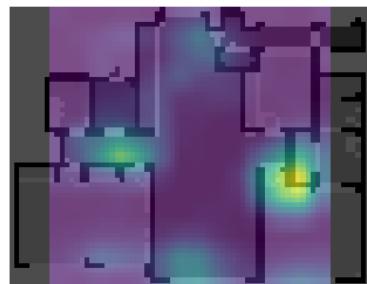


(라) 넓은 전면 폭 (복도 양쪽)

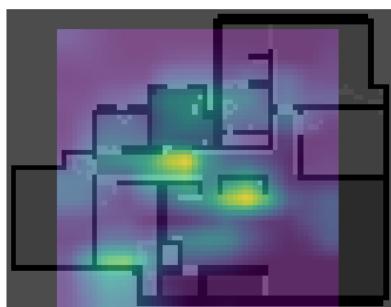
그림 5.27: 0010-2 유형의 주거공간 배치 계획 특성



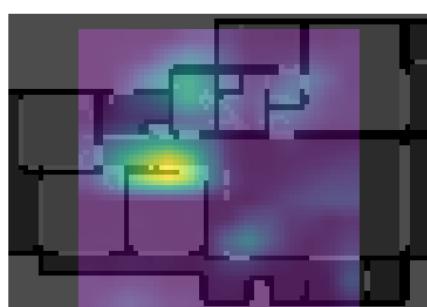
(가) 탑상형 복도, DK, 침실



(나) 넓은 전면 폭 (주 침실), 복도

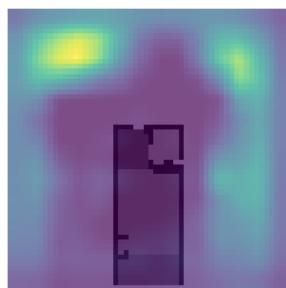


(다) 탑상형 복도, 전면 LDK

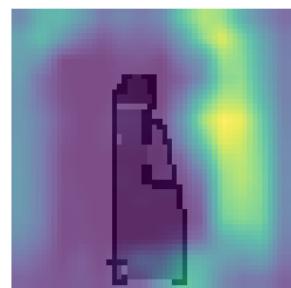


(라) 현관 쪽 복도

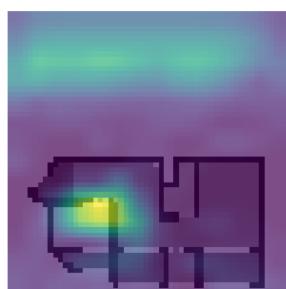
그림 5.28: 0010-3 유형의 주거공간 배치 계획 특성



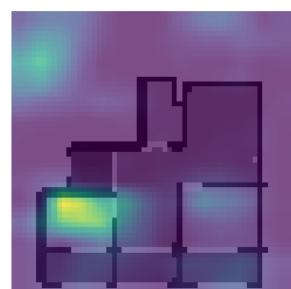
(가) 원룸형 (좁은 폭과 깊이)



(나) 원룸형 (좁은 폭과 깊이)



(다) 원룸형, 작은 침실

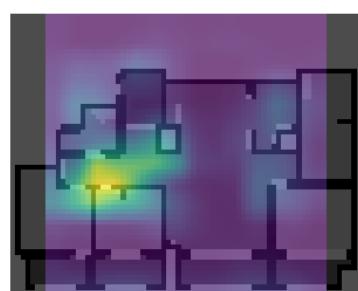


(라) 원룸형, 작은 침실

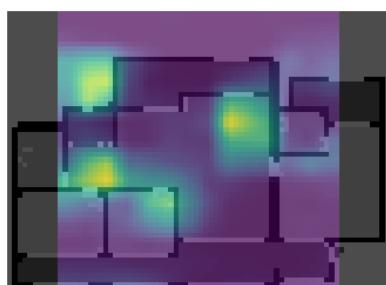
그림 5.29: 10-1 유형의 주거공간 배치 계획 특성



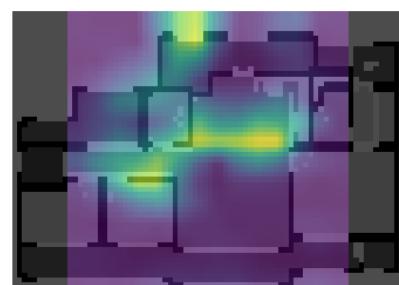
(가) 넓은 전면 폭, 좁은 침실 폭



(나) 현관 쪽 복도, 넓은 전면 폭



(다) 넓은 전면 폭, 얕은 깊이



(라) 넓은 전면 폭, 얕은 깊이

그림 5.30: 10-2 유형의 주거공간 배치 계획 특성

표 5.2: 배치유형별 계획 특성

코드	유형	특성
1	8090-1	복도형, 전면 발코니 돌출
2	8090-2	계단실형, 주 침실, 화장실, 전면 발코니 설비공간
3	8090-3	소형, 전면 발코니 베이
4	9000-1	전면 2베이 후면 3베이, 주 침실 + DK, 전면 발코니
5	9000-2	전면 전체 발코니
6	9000-3	원룸형 + 소형, 좁은 전면 폭, 전면 발코니
7	9000-4	전면 2베이 후면 3베이, 소형, 현관 + LDK
8	00-1	발코니 폭 차별화
9	00-2	발코니 폭 차별화
10	00-3	넓은 발코니 폭
11	00-4	대형, 넓은 전면 폭
12	0010-1	전면 발코니, 넓은 전면 폭
13	0010-2	넓은 전면 폭, 현관 쪽 복도, 후면 주 침실 화장실
14	0010-3	탑상형, 넓은 전면 폭, 현관 쪽 복도
15	10-1	원룸형 도시형생활주택, 좁은 폭과 깊이, 작은 침실
16	10-2	넓은 전면 폭, 얕은 깊이, 현관 쪽 복도, 작은 침실

## 제 6 장

# 주거공간 배치유형을 통한 한국 아파트 단위평면의 진화과정 해석

이 장에서는 5장의 분석을 통하여 도출된 한국 아파트 단위평면에서 나타나는 주거공간 배치유형과 한국 아파트의 진화 과정 사이의 관계를 규명하고, 이를 통하여 이 연구에서 개발한 한국 아파트 단위평면 분석 방법론의 가능성을 검증하고자 한다.

한국 아파트의 진화는 외래 주거 형식의 토착화에서 끝나는 것이 아니라, 주거 문화, 사회경제적 배경, 아파트 관련 제도 및 정책 등의 변화에 대응하여 끊임 없이 변화해온 과정이다. 특히, 단위평면은 아파트의 물리적 환경을 구성하는 핵심 요소 중 하나이다. 따라서 단위평면에서 나타나는 주거공간 배치의 변화에서도, 한국 아파트를 둘러싼 외적 요인이 미치는 영향과 그에 대응한 아파트 계획의 변화 과정을 살펴볼 수 있다.

선행연구에서도 한국 아파트의 단위평면을 단위세대 규모 및 주동 형식 등 다양한 기준에 따라 유형화하고, 각 시기별 유형 분포를 해당 시기의 계획 경향

으로 해석하여 한국 아파트에 대한 시계열적 분석을 수행한 연구가 많았다.<sup>69</sup> 이 장에서는 선행연구에서 연역적인 기준에 따라 도출된 단위평면 유형과, 이 연구에서 귀납적 기계학습 방법론을 적용하여 도출한 주거공간 배치유형을 비교한다.

우선, 선행연구에서 제시된 단위평면 유형과 5장에서 도출된 배치유형을 시기별로 비교하여 분석하였다. 각 시기별 대표적 평면 유형 사례에 BAM 시각화를 적용하여, 각 평면에서 해당 시기의 배치유형별 모형을 활성화하는 영역을 분석하였다. 각 배치유형은 유형별 모형을 가장 잘 활성화하는 단위평면의 집합에서 귀납적으로 도출되므로, 활성화 영역의 배치를 통하여 해당 유형별 모형이 인식하는 주거공간의 배치 특성을 설명할 수 있다. 이러한 분석을 각 시기별로 수행하여, 각 배치유형이 해당 시기 한국 아파트 계획의 어떠한 변화를 드러내는지 해석하였다. 이러한 과정을 통하여, 딥 러닝을 활용한 귀납적 분석 방법론이 건축 연구에 활용될 수 있는 가능성을 검증하고자 하였다.

5.2절에서 도출된 주거공간 배치유형은 딥 러닝 모형이 학습한 단위평면의 배치 계획을 그 유사성에 따라 나누어 유형화한 것이다. 딥 러닝 모형이 찾아낸 배치 특성은 모형을 구성하는 신경망 연결 구조에 저장된다. 딥 러닝 모형의 입력 계층에서 출력 계층까지 이어지는 신경망 연결의 가중치는, 기계학습 과정을 통하여, 단위평면의 각 부분과 시기별 배치 특성 사이의 관계를 재현하게 된다. 그 중에서도, 출력 계층의 직전 계층인 잠재표현 계층은 단위평면 전반에 걸친 배치 특성을 종합적으로 인식한 결과를 나타내게 된다.

이렇게 단위평면의 주거공간 배치를 인식하는 잠재표현 계층은 서로 독립적으로 다른 주거공간 배치 특성을 인식하는 여러 노드로 이루어져 있다. 입력 계층에 단위평면이 입력되면, 해당 평면의 배치와 일치하는 배치 특성을 인식하는 잠재표현 계층의 노드는 높은 값을 가지게 되는데, 이러한 상태를 활성화된다

---

<sup>69</sup> 3.2절 p. 38에서 예로 든 것처럼, 건축계획 분야에서 다양한 주제의 연구가 시기 구분과 단위평면 유형화를 통하여 이루어져 왔다.

고 표현한다.<sup>70</sup> 딥 러닝 모형이 단위평면 데이터셋에서 찾아낸 한국 아파트의 배치 특성은 각 단위평면에 대하여 잠재표현 계층의 각 노드가 활성화되는 관계를 통하여 분석할 수 있다.

이 연구에서는 잠재표현 계층에서 인식되는 주거공간의 배치 특성을 분석하기 위하여, 단위평면과 잠재표현 계층의 노드를 짹지어 대응시켜 유형화한 주거 공간 배치유형을 도출하였다. 주거공간 배치유형은 바이클러스터링을 통하여 유사한 배치 특성을 보이는 단위평면과 그러한 주거공간 배치를 인식하고 활성화되는 잠재표현 노드를 동시에 군집화한 것이다. 따라서 각 배치유형에는 단위평면 데이터셋의 일부인 유형별 평면과, 잠재표현 계층의 일부인 유형별 모형이 포함된다. 이렇게 짹지어진 단위평면과 잠재표현 노드를 대상으로, 5.3절에서는 배치유형별 BAM 시각화를 수행하여 각 배치유형에 해당하는 계획 특성을 분석하였다.

이 연구에서 적용한 바이클러스터링 알고리즘인 spectral co-clustering은, 한 단위평면 사례는 한 개의 군집에 속한 잠재표현 노드만 활성화시키고, 반대로 한 잠재표현 노드는 한 군집의 단위평면 사례에서만 활성화된다는 가정에 따라 바이클러스터링을 수행하는 알고리즘이다. 따라서 spectral co-clustering을 적용하여 도출된 배치유형을 구성하는 유형별 평면과 유형별 모형 사이에서는 1:1 관계가 성립하게 된다. 그러나 5장에서 살펴본 것처럼, 실제 분석 결과에서는 이러한 1:1 관계가 나타나지 않았다.<sup>71</sup> 이러한 배치유형의 특성을 종합적으로 이해하기 위해서는 서로 다른 유형에 속한 평면과 모형 사이의 관계까지

<sup>70</sup>딥 러닝 모형을 구성하는 신경망의 각 노드는 입력 계층의 값에 따라 달라지는 값을 갖는다. 신경망 연결의 가중치는 노드의 값과 곱해지게 되므로, 노드의 값이 0보다 커질수록 연결된 다음 계층 노드의 값에 미치는 영향력이 커지게 된다. 잠재표현 계층의 각 노드는 출력 계층의 각 노드에 연결되어 있으므로, ‘활성화된’ 잠재표현 노드는 해당 특성이 나타나는 단위평면에 대한 정보를 입력된 단위평면의 준공 시기 예측에 더 강하게 반영한다.

<sup>71</sup>5.2절에서 논의한 바와 같이, 그림 5.7에 표현된 각 유형별 평면에 대한 유형별 모형의 평균 활성화 강도에서는 1:1 관계에 한정되지 않는 결과가 나타났다. 5.3절의 유형별 배치 특성 분석은 동일한 유형에 해당하는 평면과 모형의 관계를 분석한 것으로, 1:1 관계에 해당하는 관계에 대해서만 설명이 가능하고고, 배치유형별 모형의 나머지 특성은 설명할 수 없었다.

포함한 전반적인 분석이 필요하다.

따라서, 이 장에서는 한국 아파트의 다양한 조건에서 나타나는 단위평면 사례와 해당 시기의 배치유형 모두에 대한 관계를 종합적으로 분석하였다. 분석 대상 단위평면은, 5.3절에서 BAM 시각화를 위하여 도출된 각 배치유형별 단위평면 사례 중에서, 선행연구에서 평면 계획 유형의 기준으로 삼는 주동 및 동선 형식, 평면 규모 등을 고려하여 선정하였다. 이러한 과정에 따라 도출된 분석 대상 사례는 표 6.1과 같다.

표 6.1: 분석 대상 한국 아파트 단위평면 사례

번호	주동 형식	동선 형식	전용면적 ( $m^2$ )	준공연도	지역
1	판상형	복도형	73.08	1986	서울
2	판상형	복도형	106.22	1981	서울
3	판상형	계단실형	84.82	1994	천안
4	판상형	계단실형	99.42	1994	인천
5	판상형	계단실형	84.58	2005	양주
6	탑상형	중앙부	84.40	2014	서울
7	탑상형	단부	84.25	2016	천안
8	혼합형	계단실형	79.59	2014	세종
9	도시형	복도형	23.62	2012	서울

다양한 주동 및 동선 형식별로 선정된 분석 대상 중 가장 앞선 시기에 해당하는 사례는 1980년대 판상형 주동의 복도형 동선에서 나타나는 중형 및 대형 단위평면이다. 이를 평면에 대하여 같은 시기의 계획 경향에 해당하는 8090의

3개 유형별로 BAM 시각화를 적용한 결과는 그림 6.1, 6.2와 같다.

BAM 시각화에서 밝은 노란색으로 나타나는 활성화 영역은 단위평면과 각 배치유형 사이의 연관 영역을 나타낸다. 예를 들어, 그림 6.1의 8090-1 유형 BAM 시각화 결과는, 단위평면 왼쪽<sup>72</sup> 가운데에 위치한 화장실에서 가장 밝게 활성화되는 것을 보여준다. 이는 8090-1 유형의 잠재표현 노드가 측면 중앙에 위치한 화장실과 그 주변에서 나타나는 주거공간 배치에 가장 강하게 반응하여 해당 배치유형으로 인식함을 의미한다. 주거공간 배치에 대한 반응이 크고 해당 유형에 더 정확하게 일치할수록, BAM 시각화에서 활성화가 강하게 나타나는 영역과 나머지 영역의 대비가 높아지게 된다.

판상형 주동의 복도형 단위평면에 대한 BAM 시각화 결과에서, 8090-1과 8090-3 유형은 현관에 가까운 측면 중앙에 배치된 화장실과 강한 연관을 나타내었다 (그림 6.1, 6.2). 8090-2 유형은 국민주택 규모 중형 평면에서 현관과 먼 쪽에 배치된 거실과 발코니의 경계 영역에서 강한 활성화를 보였다 (그림 6.1).

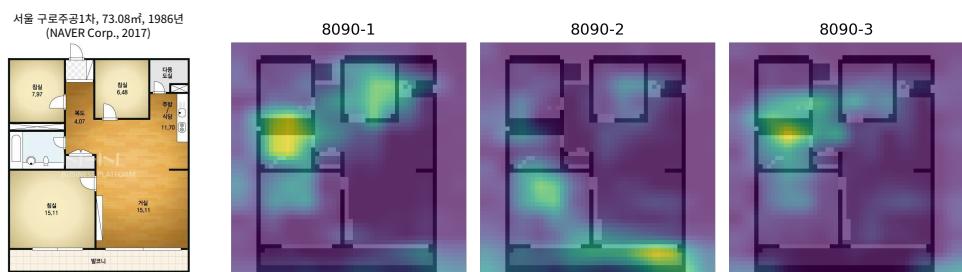


그림 6.1: 복도형 중형 평면에 대한 유형별 모형의 활성화 영역

한국 아파트에서, 판상형 주동의 계단실형 동선 계획은 1980년대에 이미 규범

<sup>72</sup>4.3.1절에서 데이터셋의 주거공간 배치 데이터를 현관이 왼쪽에 위치하도록 정규화하였으므로, 단위평면의 왼쪽은 양 측면 중 현관에 가까운 쪽을 의미한다.

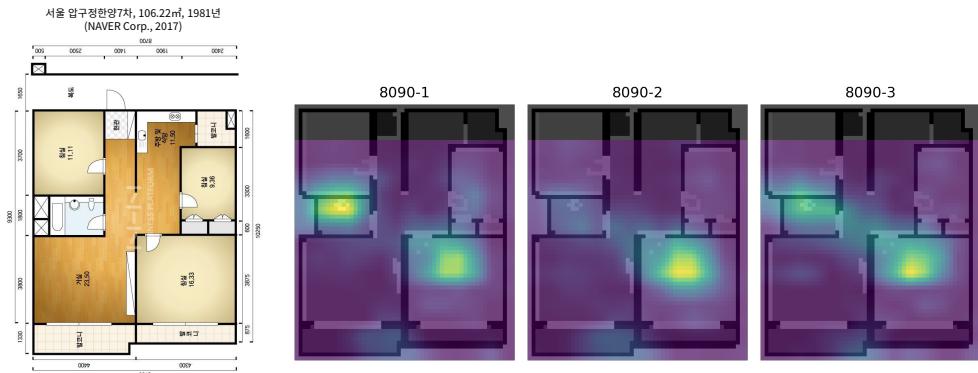


그림 6.2: 복도형 대형 평면에 대한 유형별 모형의 활성화 영역

으로 자리잡았다.<sup>73</sup><sup>74</sup> 측면 중앙에 배치된 현관을 통하여 측면에서 단위세대로 진입하는 1990년대 전형적인 계단실형 단위평면 (그림 6.3, 6.4)과 비교할 때, 계단실형 단위평면의 현관 위치에 화장실이 배치된 것은 이후 시기의 단위평면 사례 대부분에 해당하는 계단실형 단위평면에 대하여 복도형 단위평면을 구분하기 위한 가장 큰 차이점이 된다. 따라서 복도형 주동의 단위평면에서 나타나는 주거공간 배치는 계획 시기를 1980년대 이전으로 특정하기 위한 좋은 단서가 될 수 있다.

한편, 8090 3개 배치유형 모두에서 현관 반대쪽 전면에 배치된 주 침실을 주요한 주거공간 배치 특성으로 지목하였다 (그림 6.2). 이러한 주거공간 배치는 복도형 주동에서만 나타난 것은 아니다. 1990년대 전형적인 계단실형 단위평면 (그림 6.3, 6.4)에서도 현관에서 면 쪽 전면에 주 침실이 배치되어 있으며, 8090 배치유형의 모형은 그 평면들에서도 이러한 주 침실의 배치를 인식하여

<sup>73</sup> 당시 주류였던 판상형 주동에 한하여, 동선 계획을 기준으로 유형을 나누어 분석하였다.

“아파트 주동의 형식을 계단실형, 편복도형, 탑상형, 복층형의 네 가지로 분류하여 … 계단실형 아파트가 아파트 계획의 규범으로 자리 잡고 있음을 확인할 수 있다.” (최재필 등. (2004). 국내 아파트 단위주호 평면의 공간 분석-1966 년 2002 년의 서울지역 아파트를 대상으로. 대한건축학회 논문집-계획계, 20(6), 153-162)

<sup>74</sup> 최재필 등. (2004). 국내 아파트 단위주호 평면의 공간 분석-1966 년 2002 년의 서울지역 아파트를 대상으로.

활성화되었다. 따라서 8090 3개 배치유형이 곧 복도형 동선 계획만을 인식하거나 그러한 계획 경향만을 의미하지는 않음을 알 수 있다.

이러한 특성은 딥 러닝 모형이 학습한 한국 아파트 단위평면 데이터셋의 특성에서 그 원인을 찾을 수 있다. 이 연구에서 활용한 딥 러닝과 같은 기계학습은 귀납적 방법론으로, 데이터셋에 내재된 규칙을 발견할 수 있지만, 반대로 데이터가 충분하지 않은 경우는 분석이 어렵다. 4장에서 구축한 한국 아파트 단위평면 데이터셋은 현존하는 아파트 재고 사례를 수집한 것으로, 신도시 개발과 아파트 건설이 본격화된 1990년대 이후 사례가 93.7%에 달한다. 따라서 이 연구에서 도출된 주거공간 배치유형도 1990년대 이후의 진화과정에 집중되었다.

5장의 분석 결과에서 8090 3개 배치유형은 딥 러닝 모형에서 1990년대 이전의 단위평면에서 나타나는 주거공간 배치를 인식하는 부분을 대표하지만, 이 유형에 해당하는 단위평면 사례는 1990년대 이후까지 걸쳐 나타났다 (그림 5.9, 1번-3번 군집). 마찬가지로, 복도형과 계단실형 단위평면 사례 모두 1980년대와 1990년대에 걸쳐 등장하였다. 따라서 8090 3개 배치유형은, 복도형과 계단실형 동선 계획을 아울러, 1990년대 이전부터 이어진 단위평면 계획에서 나타나는 주거공간의 전반적인 배치를 종합적으로 유형화한 것으로 해석할 수 있다.

복도형 단위평면과 8090 배치유형에 대한 분석 결과를 종합하면, 복도형 평면에 대하여 준공 시기만으로 해당 시기의 배치 특성을 찾도록 한 결과, 복도형 평면 계획의 다양한 특징 중에서 이후 시기의 계단실형 평면과 구분하기 위하여 필요한 가장 확실한 차이점에 집중하였음을 알 수 있다.

1990년대 계단실형 주동의 중형 및 대형 단위평면에 대한 BAM 시작화 결과에서, 9000 4개 배치유형은 1990년대 계단실형 평면에서 나타나는 여러 주거 공간의 배치를 각각 독립적으로 인식하였다 (그림 6.3, 6.4). 그림 6.3의 국민주

택 규모 3LDK 단위평면에서, 9000 배치유형은 전면 2베이 후면 3베이 배치와 현관 반대쪽 전면에 위치한 주 침실과 후면 중앙에 계획된 DK (9000-1,4), 전면 전체에 걸친 발코니 (9000-2), 계단실형 동선의 측면 진입으로 현관 반대편으로 옮겨간 화장실 (9000-3) 등 단위평면의 각 공간을 구체적으로 인식하였다. 그림 6.4의 대형 4LDK 단위평면에서 9000 배치유형 중 3개 유형은 후면 3베이 배치에서 거실과 DK 사이에 공용 화장실을 배치하면서 나타난 베이 내부 공간의 분할에 집중하였다 (9000-1,3,4). 8090 배치유형 중 일부에서도 4LDK 평면에서 현관 쪽 후면 침실과 그에 인접하여 배치된 공용 화장실을 이 시기 계단실형 단위평면의 배치 특성으로 도출하였다. (8090-1,3).

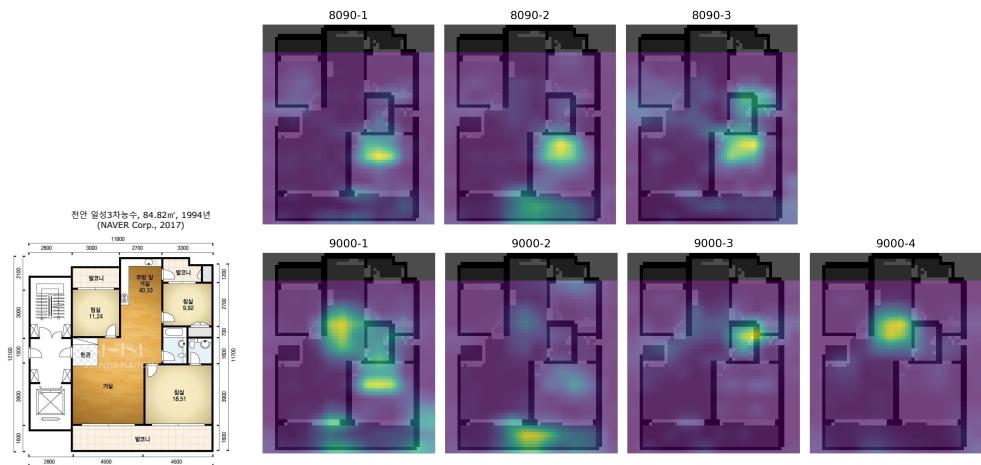


그림 6.3: 계단실형 중형 평면에 대한 유형별 모형의 활성화 영역

1990년대 계단실형 3LDK 단위평면에서 다양한 주거공간이 그 시기의 계획 경향으로 고르게 인식되는 것은, 해당 단위평면 사례에서 여러 주거공간의 배치 계획이 일관되게 나타나는 경향이 학습된 것으로 해석할 수 있다. 계단실이 전후면 전체에 걸쳐 계획된 판상형 주동이라는 조건에서, 침실 3개 및 화장실 2개를 갖춘 단위평면을 계획하면서 여러 공간의 배치가 서로 맞물려 유사하게 나타났다는 의미이다.

선행연구에서도 1990년대에 들어서면서 한국 아파트 단위평면이 소수 유형으

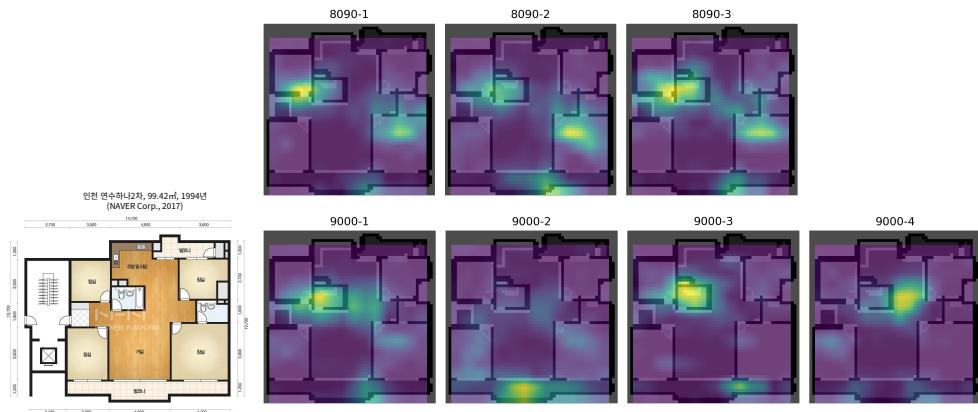


그림 6.4: 계단실형 대형 평면에 대한 유형별 모형의 활성화 영역

로 수렴하는 과정을 확인하였다.<sup>75</sup> 계단실형 코어 배치를 통하여 전후면 전체가 개방되고, 거실, 식당, 주방이 통합된 LDK가 단위세대 중앙에 배치되어,<sup>76</sup> 동선 공간으로서의 중앙 홀의 역할을 겸하는 계획 경향이 확립되었다.<sup>77</sup> 이러한 1990년대 한국 아파트의 전형적인 배치 계획은 이 연구에서 귀납적으로 발견한 계단실형 단위평면의 계획 경향과 일치한다. 이는 이 연구의 방법론을 적용하여 단위평면의 계획 특성을 확인할 수 있음을 의미한다.

한편, 1990년대 계단실형 대형 4LDK 단위평면에서 나타나는 전면 3베이 배치는 해당 시기 단위평면의 계획 특성으로 인식되지 않았다 (그림 6.4). 이는 전면 3베이 배치가 다른 시기 단위평면에서도 나타나는, 해당 시기로 특정하기 어려운 특성이라는 의미로 해석할 수 있다. 실제로 주 침실에 부속된 드레스

<sup>75</sup>Seo. (2007). Space puzzle in a concrete box: finding design competence that generates the modern apartment houses in Seoul. Environment and Planning B: Planning and Design, 34(6), 1071–1084.; 최재필 등. (2004). 국내 아파트 단위주호 평면의 공간 분석-1966 년 ~ 2002 년의 서울지역 아파트를 대상으로. 대한건축학회 논문집-계획계, 20(6), 153–162.

<sup>76</sup>도연정, 전봉희. (2017). 한국 아파트 평면에서 LDK 형성과정의 특성: 1962년~1988년 대 한주택공사 아파트 사례를 중심으로. 대한건축학회 논문집-계획계, 33(9), 61–70.

<sup>77</sup>Seo. (2007). Space puzzle in a concrete box: finding design competence that generates the modern apartment houses in Seoul.

룸을 제외하면 2000년대 중형 평면에서 계단실 전면에 침실이 계획된 전면 3베이 배치는 1990년대 대형 평면과 크게 차이가 없다 (그림 6.5). 이렇게 한국 아파트에서 대형 평면의 주거공간 배치가 시간이 지나며 소형 평면에도 적용되는 ‘하향화 (trickle-down)’ 현상은 1990년대에 이미 확인된 바 있다.<sup>78</sup><sup>79</sup> 따라서, 1990년대 대형 평면에서 9000 배치유형은 이후 시기의 중소형 평면과의 구분에 도움이 되지 않는 전면 3베이 배치를 무시하고, 나머지 영역에 집중하였음을 알 수 있다.

그림 6.5의 2000년대 중형 단위평면 사례에 대한 BAM 시각화 결과에서는, 2000년대 평면에 대응되는 00 4개 배치유형의 모형 모두가 전면 발코니에서만 집중되어 활성화되었다. 00-1과 00-2 유형의 모형은 중앙 베이의 거실 발코니가 후면 계단실의 전면에 계획된 침실 발코니보다 돌출되어 생겨난 외부 공간에 집중하였다. 한편, 00-3 유형의 모형은 전면 발코니와 배후 실의 경계를 중심으로 활성화되었다. 주거공간 배치 데이터셋에서 단위평면은 분석 영역의 하단에 정렬되므로, 전면 발코니의 배후 경계는 전면 발코니의 폭에 따라 그 위치가 달라지게 된다. 따라서, 2000년대 평면에서 전면 발코니의 배후 경계에 집중하는 것은 그 경계가 전면 발코니의 넓은 폭 때문에 후퇴하여 표현된 것을 인식한 것으로 해석할 수 있다. 00-4 유형의 모형은 배후 실과의 경계와 더불어, 분석 영역의 가장자리까지 이어지는 전면 발코니의 측면 경계를 함께 인식하였다. 이는 계단실 후면 배치와 함께 단위평면의 폭이 늘어난 것을 전면 발코니를 통해 인식한 것으로 해석된다. 종합하면, 2000년대에 한정되어 나타났던 계획 경향에 해당하는 00 배치유형 4개 모두에서 유형별 모형이 넓은 발코니 폭을

<sup>78</sup> “평면형 및 계획요소의 하향화 (trickle-down) 현상은 대규모 평형에 적용되었던 평면형이나 계획 및 설계요소가 점차 그보다 적은[원문대로] 규모의 평면에 영향을 미쳐 … 소규모 평형에 그대로 적용되는 현상을 말한다.” (김수암, 김상호. (1997). 우리나라 민간아파트 주호평면의 시계열적 흐름에 관한 연구. 주택연구, 9, 103-127)

<sup>79</sup> 김수암, 김상호. (1997). 우리나라 민간아파트 주호평면의 시계열적 흐름에 관한 연구; 최재필. (1996). 공간구문론을 사용한 국내 아파트 단위주호 평면의 시계열적 분석-수도권 4LDK 아파트 단위주호 평면계획을 중심으로. 대한건축학회 논문집, 12(7), 15-27.

집중하여 인식하였다.

이는 2000년대에 발코니 바닥면적 산정 기준이 가장 완화되었던 것과 관련이 있다. 발코니는 이미 1986년부터 1.2 m 깊이까지 바닥면적 산입에서 제외되었으나, 2000년에는 간이화단을 설치하면 2 m 깊이까지 제외하도록 기준이 완화되었다. 그러나 간이화단을 통한 건축물 녹화라는 목적은 거의 달성되지 않았고, 발코니에 창호를 설치하여 실내 공간으로 전용되는 경우가 많아, 2005년에 폐지되었다.<sup>80</sup> 따라서 2 m 깊이의 발코니는 2000년대 전반에 계획 및 허가된, 준공연도 기준으로는 2000년대의 아파트 단위평면에서만 나타날 수 있는 특징이 될 수 있다. 이렇게 2000년대와 2 m 발코니처럼 분류 기준과 주거공간 배치 사이에 명확한 연관관계가 있을 때, 딥 러닝 분석을 통하여 그러한 계획 특성을 찾아낼 수 있음을 확인할 수 있다.

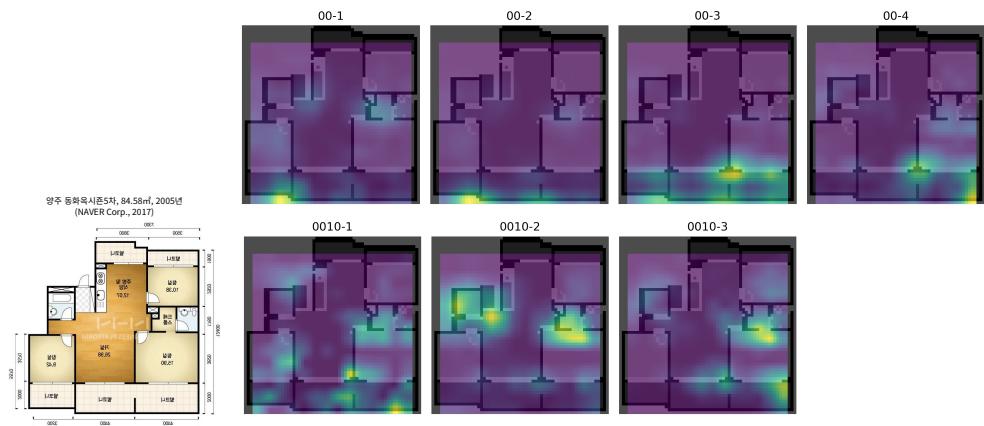


그림 6.5: 2000년대 계단실형 평면에 대한 유형별 모형의 활성화 영역

2010년대까지 이어지는 계획 경향에 해당하는 0010 유형과 10 유형은, 지금 까지 살펴본 배치유형들과는 달리, 해당 계획 경향의 종결 시점을 단정할 수 없

<sup>80</sup>권혁삼 등. (2006). 공동주택 발코니 공간의 효율적 활용방안 연구. 대한건축학회 학술발표 대회 논문집-계획계, 26(1), 81–84.; 박인석 등. (2014). 전용면적 산정기준 변화와 발코니 용도변환 허용이 아파트 단위주거 평면설계에 미친 영향: 전용면적 60m<sup>2</sup> 와 85m<sup>2</sup> 평면의 실별 규모 변화를 중심으로. 한국주거학회논문집, 25(2), 27–36.

다. 8090 유형부터 00 유형까지의 배치유형은 2000년대 이전까지 나타난 계획 경향에 해당한다. 단위평면 데이터셋은 2010년대 단위평면 사례를 포함하고 있으므로, 유형별 평면 사례의 준공 시기를 통하여 이들 배치유형이 2010년대 이후로 이어지지 않는 것을 확인할 수 있다.

그러나 0010 유형과 10 유형은 2010년대 사례를 포함한 계획 경향이기 때문에, 종결 시점을 파악하기 위해서는 그보다 이후 시기에 대한 정보가 필요하다. 단위평면 데이터셋에 포함된 아파트 사례는 2017년 수집된 것으로, 준공연도 기준 2020년까지의 사례를 포함하고 있다. 수집 시점 이후가 절반 이상에 해당하는 2010년대 후반은 수집된 단위평면이 이 시기 한국 아파트 전수를 대표한다고 기대할 수 없다. 따라서 2010년대 전반까지 이어진 배치유형은 해당 계획 경향의 변화 및 종결 시점을 확인하기 어렵다.

이러한 점을 고려하여, 2000년대부터 2010년대 사례에 해당하는 0010 유형은 2000년대에 새롭게 등장한 배치유형으로만 해석한다. 마찬가지로 10 유형은 2010년대에만 등장하는 것이 아니라, 2010년대 시작된 계획 경향을 의미한다.

그림 6.5의 2000년대 계단실형 단위평면에 대한 유형별 모형의 BAM 시각화 결과에서, 0010 유형은 1990년대까지의 전형적인 계단실형 평면에서는 나타나지 않았던 새로운 주거공간 배치를 찾아내었다. 주 침실에 드레스룸이 부속되고 (0010-2,3), 계단실이 후면으로 돌출하면서 후면에서 전면으로 진입하는 현관이 안쪽으로 배치되고, 공용 화장실은 바깥쪽으로 배치되는 계획 (0010-2) 등이다.

판상형 주동이 아닌 탑상형 주동에서는 단위평면의 외부 조건이 크게 변화함에 따라 주거공간 배치의 변화가 더 다양하게 나타났다. 탑상형 주동의 코어 남쪽에 계획된 중앙부 단위평면에서는 두 단위세대가 후면을 서로 맞대어 계획되기 때문에, 전면과 측면이 외기에 개방되었다. 이러한 조건은 단위세대가 일렬로 계획되어 전후면이 개방된 판상형 주동과 크게 다른 주거공간 배치가 나타날

수 밖에 없게 하였다.

이렇게 주동 형태라는 하나의 요인에 의하여 여러 변화가 동시에 일어날 때, 복도형 단위평면에 대한 분석 결과에서와 마찬가지로, 딥 러닝 모형은 다양한 변화를 모두 확인하기보다는 시기 구분이라는 목표 달성에 가장 크게 도움이 되는, 가장 확실하게 구분할 수 있는 차이점을 찾는 모습을 보였다. 탑상형 중앙부 단위평면에서 0010 유형이 찾아낸 차이점은 LDK가 더 이상 동선 공간으로 기능하지 않는 배치였다.

판상형 주동의 단위평면에서 LDK는 단위평면 중앙에 배치되어 전면부터 후면까지 하나의 공간으로 통합되어 계획되었다. 이러한 단위평면에서 LDK는 단위세대 전체 동선을 연계하는 중정 혹은 중앙 홀의 역할을 겸하게 되었다.<sup>81</sup>

그러나 탑상형 중앙부 평면에서는 후면에 인접 세대가 배치되면서 외기 노출이 불가능하게 되고, 이에 따라 DK가 후면이 아닌 전면에 배치되었다. 이러한 배치로 인하여, LDK는 더 이상 동선을 연계하는 역할을 맡지 않게 된다. 한편, 4베이 이상으로 계획된 전면 폭의 증가로 평면 전체를 가로지르는 복도가 계획된다. 그림 6.6에서 0010 3개 유형의 모형은 평면 전체를 가로지르는 복도와 동선 공간과 벽으로 구획된 전면 배치 DK를 인식하여 활성화되었다.

또한, 모든 단위평면이 같은 외부 조건을 공유하는 판상형 주동과 달리, 탑상형 주동에서는 다양한 외부 조건이 나타난다. 1990년대 계단실형 단위평면과 마찬가지로 측면에서 진입하는 탑상형 주동의 단부 단위평면이나, 물리적으로는 탑상형 주동이지만 계단실형 동선 계획은 판상형 주동과 같은 L자형 혹은 혼합형 주동의 단위평면에서는 판상형 주동에서와 마찬가지로 전면과 후면이

<sup>81</sup>Seo. (2007). Space puzzle in a concrete box: finding design competence that generates the modern apartment houses in Seoul. Environment and Planning B: Planning and Design, 34(6), 1071-1084.; 최재필 등. (2004). 국내 아파트 단위주호 평면의 공간 분석-1966 년 2002 년의 서울지역 아파트를 대상으로. 대한건축학회 논문집-계획계, 20(6), 153-162.

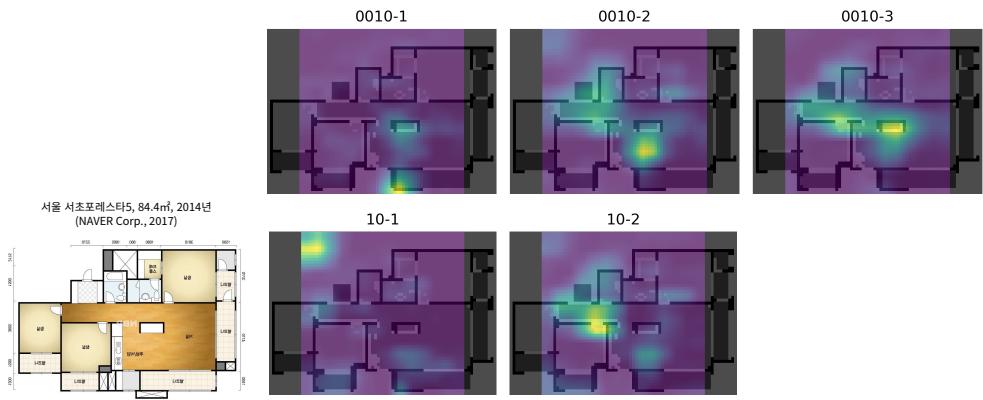


그림 6.6: 탑상형 중앙부 평면에 대한 유형별 모형의 활성화 영역

외기에 개방된다.

탑상형 단부 단위평면은 한쪽 측면은 외기에 노출되므로 3면이 개방되도록 계획될 수 있다. 그러나, 측면이 개방된 경우에도 발코니 확장 허용 이후 전면 폭이 더욱 늘어나면서 그림 6.6의 경우처럼 개방면과 그에 면한 발코니가 분석 영역 밖에 위치하게 된다. 그렇기 때문에, 측면 개방 여부는 분석 영역 안의 주거공간 배치 분석에는 직접적인 영향을 미치지 못한다.<sup>82</sup> 이러한 점을 고려하여, 2010년대 단위평면에 대한 배치유형의 활성화 영역 분석에는 측면이 개방되지 않은 단위평면을 적용하였다. 그림 6.7, 6.8에서 볼 수 있듯이, 탑상형 단부와 혼합형 주동 평면 사례에서는 측면 개방 없이 계단실형 단위평면과 동일한 외기 노출 조건을 보였다.

이러한 단위평면에서, LDK는 여전히 중앙에 위치하며 동선 공간으로서의 기능을 겸하고 있다. 그러나 전면 폭의 증가와 함께 여전히 평면을 가로지르는 복도는 여기서도 등장하였다. 0010-3 유형은 탑상형 단부 단위평면에서 현관과 LDK 사이에 계획된 복도를 인식하였다.

<sup>82</sup>분석 영역 안에 포함되지 않는 측면 개방도 그러한 계획이 중앙부의 주거공간 배치에 미치는 영향을 통해서 간접적으로는 확인할 수 있다.

0010-1 유형의 모형은 해당 유형의 평면에 대해서는 전면 발코니에서 나타나는 베이의 위치와 관련된 영역에서 활성화되었다. 베이의 중앙 또는 벽체 위치를 통하여, 분석 영역 바깥쪽으로 밀려 지나치게 좁게 나타나거나, 발코니 확장의 영향으로 실제로 좁은 폭으로 계획된 베이를 인식하였다. 이러한 특성은 그림 6.7의 탑상형 단부 평면에서도 잘 나타난다. 한편, 그림 6.8의 혼합형 주동 평면에 대해서는 후면 발코니의 경계를 통하여 동일한 규모에서 종횡비가 늘어나면서 평면의 깊이가 얕아지는 것을 인식하였다.

0010-2 유형과 0010-3 유형의 모형은 전체 단위평면에 대한 활성화 강도의 상관계수가 높게 나타났다 (그림 5.8, 13번과 14번). 이는 동일한 평면에서 0010-2 유형과 0010-3 유형의 모형이 함께 활성화되는 경향이 높다는 의미이다. 두 배치유형은 복도나 전면 DK 배치와 같은 탑상형 주동 평면의 특징을 인식하였지만, 0010-1 유형과 함께 평면 깊이의 감소로 인한 후면 발코니 배치도 인식하였다. 그림 6.7, 6.8에서 두 유형의 모형은 후면에 배치된 공용 화장실과 주 침실 화장실, 후면 중앙 주방 및 식당 등이 후면 발코니와 인접한 영역에서 활성화되었다.

2010년대 소형 이상 (원룸형 제외) 단위평면에서 새롭게 등장한 계획 경향을 나타내는 10-2 유형은 높은 종횡비의 단위평면에서 나타나는 주거공간의 여러 배치 특성을 더 구체적으로 인식하였다. 깊이가 얕은 침실 (그림 6.6), 폭이 넓은 LDK와 복도, 얕은 평면 깊이 (그림 6.7), 계단실 전면 침실 및 현관, LDK 등의 얕은 깊이 (그림 6.8) 등이었다.

0010 3개 평면유형과 10-2 유형이 다양한 주동 형태의 단위평면에 대하여 넓은 전면 폭과 얕은 평면 깊이를 해당 시기 단위평면의 배치 특성으로 인식한 것은, 단위평면의 외부 조건 차이를 넘어서는 일관된 계획 경향을 인식한 것으로 해석할 수 있다. 특히, 같은 면적에서 전면 폭은 증가하고 깊이는 감소하는 것은, 2000년대 이후 발코니 면적을 극대화하여 실내 면적을 증가시키는 계획 경향을 지칭한다. 2006년부터 발코니 확장이 허용되었으나, 허용 이전에도

이미 발코니 확장은 2004년 견본주택에서 발코니를 확장하는 것을 금지할 정도로 일반화되어 있었다.<sup>83</sup> 따라서 2000년대 후반 서울 강남 3개 구 아파트에서 전용면적 85 m<sup>2</sup> 평면의 발코니 면적은 1980년대 후반 16.55 m<sup>2</sup>에서 2000년대 후반 이후 30.11 m<sup>2</sup>로 2배 가까이 증가하였다.<sup>84</sup>

발코니 확장이 전제가 된 이후, 발코니 면적에서 얻어지는 실내 면적을 극대화 하지 못하는 공간 배치는 경제적으로도 불리한 위치에 놓이게 되었다. 따라서 2010년대에는 베이 폭이 좁고 깊이가 얕은, 작은 침실이 계획 특성이 될 정도로, 발코니를 확장하지 않으면 거주가 어려울 정도의 평면이 등장하게 되었다. 이러한 경향은 시간이 지남에 따라 더욱 강화되어 이제는 이 연구에서 분석한 실제 건설되는 아파트 뿐만 아니라 현상설계 당선안에서도 발코니 확장을 전제한 평면 계획이 당연시되고 있다.<sup>85</sup>

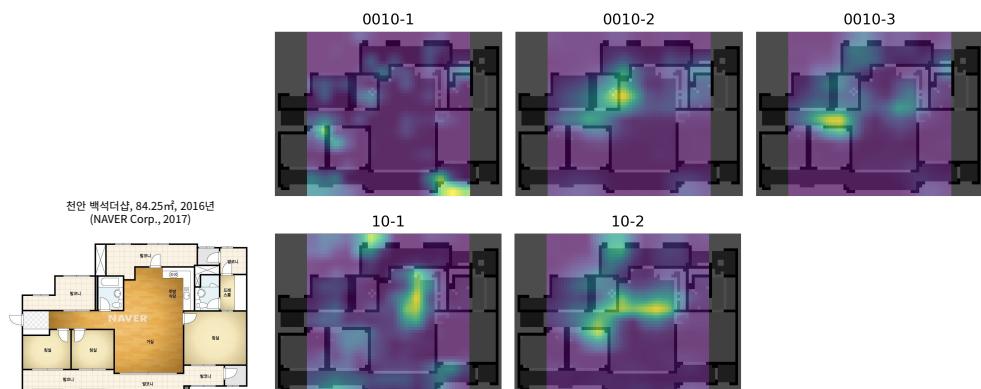


그림 6.7: 탑상형 단부 평면에 대한 유형별 모형의 활성화 영역

10-1 유형은 2009년 등장한 원룸형 도시형생활주택에 해당한다. 이와 더불어

<sup>83</sup>권혁삼 등. (2006). 공동주택 발코니 공간의 효율적 활용방안 연구. 대한건축학회 학술발표 대회 논문집-계획계, 26(1), 81-84.

<sup>84</sup>박인석 등. (2014). 전용면적 산정기준 변화와 발코니 용도변환 허용이 아파트 단위주거 평면설계에 미친 영향: 전용면적 60m<sup>2</sup> 와 85m<sup>2</sup> 평면의 실별 규모 변화를 중심으로. 한국주거학회논문집, 25(2), 27-36.

<sup>85</sup>윤효진. (2019). 아파트의 주동형태 및 확장형 발코니에 따른 단위세대 평면계획 변화특성. 한국퍼실리티매니지먼트학회지, 14(1), 61-69.

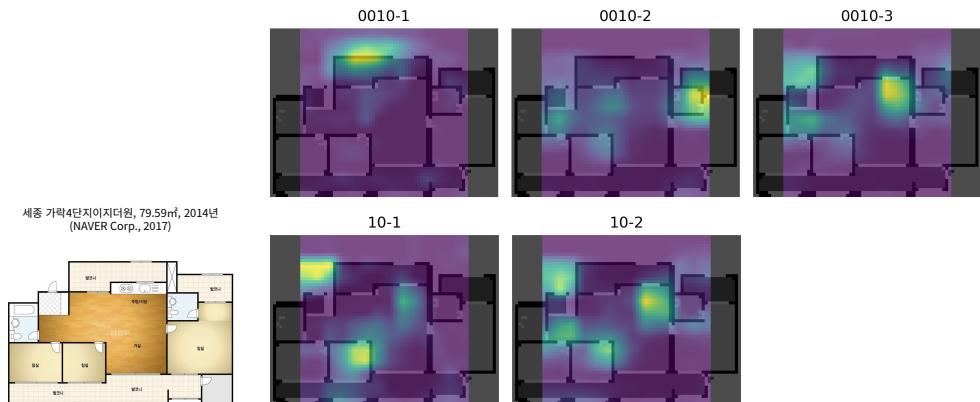


그림 6.8: 혼합형 주동 평면에 대한 유형별 모형의 활성화 영역

소형 평면이 분류된 8090-3 유형, 9000-3 유형을 함께 살펴보면, 원룸이라는 내부의 주거공간 배치보다는 주변의 빈 공간을 통하여 작은 규모 자체를 인식하고 있음을 알 수 있다.

이러한 현상은 평면의 배치를 절대적인 축척으로 비교하고자 할 때 고정된 크기의 분석 영역을 입력받는 딥 러닝 분석의 한계를 보여주었다. 분석 영역과 유사한 규모의 평면에서는 이러한 분석 방식이 적합하지만, 대형 평면과 소형 평면을 함께 절대적인 축척으로 분석하는 방법론으로서는 소형 평면의 실내 주거공간 배치 분석에 아직 부족함이 있다. 이를 개선하여 대형 평면의 전체를 분석하면서 동시에 소형 평면에서도 내부의 주거공간 배치를 분석하도록 하기 위해서는 분석 축척을 평면 규모에 맞추어 조절하면서 딥 러닝 모형이 규모 정보를 따로 입력받을 수 있도록 하는 등 추가적인 단계가 필요할 것이다.

이 장에서는 다양한 주동 형식 및 동선 형식에 따른 한국 아파트의 전형적인 단위평면 계획에 대하여 이 연구에서 도출된 주거공간 배치유형이 어떠한 주거 공간 배치를 인식하고 시기별 한국 아파트의 진화 과정을 어떻게 재구성하는지 분석하였다. 이러한 과정을 통하여, 딥 러닝 모형은 복도형 및 계단실형 동선 계획의 차이, 발코니 면적 극대화를 위한 전면 폭 및 베이 수 증가, 탑상형 주

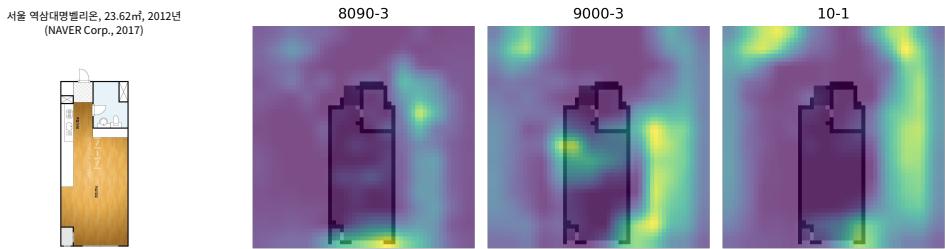


그림 6.9: 원룸형 도시형생활주택 평면에 대한 유형별 모형의 활성화 영역

동의 등장 및 혼합형 주동으로의 절충화, 한국 아파트에서 원룸형 규모의 확대 등의 계획 경향을 통하여 한국 아파트의 진화를 인식하였음을 확인하였다.

딥 러닝 모형은 한국 아파트에서 나타나는 단위평면의 주거공간 배치를 잘 학습하였으나, 시기 구분을 통하여 학습한 주거공간 배치는 시기에 따라 가장 명확하게 구분되는 차이점에 집중하는 경향이 있었다. 그에 따라, 한국 아파트의 단위평면이 소수 유형으로 수렴한 전면 2베이 후면 3베이 3LDK 평면을 기준으로, 이전 및 이후 시기의 단위평면에서는 가장 다르게 나타나는 주거 공간 배치만을 인식하는 모습을 보였다. 특히, 소형 평면에서는 국민주택 규모로 설정된 분석 영역 안에서 외부 공간이 표현되는 것 자체를 해당 평면의 특성으로 인식하여, 단위평면 내부의 주거공간 배치에 대한 인식이 잘 이루어지지 않았다. 이 장의 분석을 통하여 이러한 문제를 확인하고, 개선하기 위한 추후 연구 방향을 논의하였다.

## 제 7 장

### 결론

이 연구는 한국 아파트에 대한 전수조사를 가능하게 하는 새로운 방법론의 필요성에서 시작되었다. 한국 아파트에 대한 연구는 한국에서 아파트가 주류 주거 유형이 된 이후로 지역, 시기, 규모, 설계 주체 등 다양한 기준으로 나누어져 파편화되었다. 이 연구에서는 개별 사례에 대한 수작업이 필요하지 않은 분석 방법론을 통하여 한국 아파트 전수에 대한 연구가 가능하도록 하고자 하였다. 이를 위하여, 딥 러닝과 기계학습 방법론을 활용하여 한국 아파트 단위평면의 주거공간 배치에서 귀납적으로 학습한 규칙을 도출할 수 있는 분석 방법론을 개발하였다. 이를 한국 아파트 전수 데이터셋에 적용하여 한국 아파트 단위평면의 진화과정을 분석하고, 방법론을 검증하였다.

2장에서는 기계학습 방법론과 그에 속한 딥 러닝 방법론을 살펴보고 건축 공간에 딥 러닝을 적용한 연구에 대한 고찰을 통하여 건축 공간 분석에 딥 러닝을 활용하기 위한 전략을 모색하였다. 기계학습 방법론은 데이터에서 귀납적으로 규칙을 도출할 수 있으며, 입력 데이터에 따라 동일한 방법론으로 다양한 대상을 분석할 수 있는 특징이 있다. 딥 러닝은 기계학습 방법론 중 하나로, 복잡한 개념을 처리할 수 있어 최근 다양한 분야에서 활용되고 있음을 확인하였다.

특히 합성곱 신경망 (CNN)은 이미지나 바둑판 등 격자 구조의 데이터를 처리하는 데 특화되어 있다. 따라서 여러 선행 연구에서 건축물의 평면을 2차원 이미지 형태로 표현하여 건축 공간에 딥 러닝을 성공적으로 적용하였다. 딥 러닝 모형이 데이터에서 학습한 규칙은 잠재공간 분석이나 역합성곱 신경망 학습 등의 방법론을 통하여 이해할 수 있다. 이러한 고찰을 통하여 건축 공간을 귀납적으로 분석하고 그 결과를 해석하기 위한 전략을 수립하였다.

3장 분석 방법론에서는 딥 러닝을 활용하여 한국 아파트 단위평면의 주거공간 배치를 귀납적으로 분석할 수 있는 방법론을 수립하였다. 건축 평면의 주거공간 배치를 3차원 행렬로 모형화하고, 이를 다양한 기준에 따라 분류하도록 딥 러닝 모형을 훈련시켰다. 그 과정에서 표현학습을 통하여 학습한 분류 범주별 주거공간 배치 특성을 CAM (class activation mapping) 방법론을 통하여 시각화하였다. 이러한 분석이 가능한 VGG-GAP 모형을 분석 모형으로 선정하였다. 이러한 분석 방법론을 검증하기 위하여, 딥 러닝 모형이 학습한 잠재 표현을 기준으로 주거공간 배치를 바이클러스터링을 통하여 유형화하였고, 그 결과를 시각화하는 BAM (bicluster activation map) 방법론을 수립하였다. 이를 한국 아파트 단위평면의 진화과정 분석에 적용하여 이 연구의 방법론을 검증할 수 있도록 하였다.

4장 단위평면 데이터셋에서는 이 연구의 방법론을 검증하기 위한 한국 아파트 단위평면 분석에 필요한 단위평면 주거공간 배치 데이터셋을 구축하였다. 한국 아파트에 대한 단위평면 등 공개 자료를 수집하고, 수집된 평면도에서 주거 공간 배치를 인식하여 처리하였다. 동일한 조건에서 주거공간 배치를 분석할 수 있도록 수집된 평면도에서 다양하게 나타나는 방향 및 축척을 정규화하였다. 딥 러닝 모형의 학습을 위해서는 입력값 전체가 의미있는 영역으로 가득 채워져야 하므로, 다양한 크기의 단위평면에 대한 분석 영역 설정 알고리즘을 수립하였다. 또한, 분류 모형의 학습에 활용할 수 있는 연관 데이터를 딥 러닝 모형에 적합하도록 처리하여, 데이터셋에 함께 구축하였다.

5장 단위평면 분석에서는 4장에서 구축한 한국 아파트 단위평면 데이터셋에 3장의 BAM 방법론을 적용하여 한국 아파트 단위평면의 진화과정을 분석하였다. 시기별 주거공간 배치 변화를 분석할 수 있도록 3장의 VGG-GAP 모형을 수정하고 4장의 데이터셋을 학습시켜 한국 아파트의 주거공간 배치를 학습한 모형을 구축하였다. 이 모형의 512개 잠재표현 노드와 5만 개 단위평면 사례를 16개 바이클러스터로 짹지어, 잠재표현에서 주거공간 배치유형을 도출하였다. 각 유형에 속한 단위평면 사례에 대하여, 시기, 지역, 규모 등 일반 특성에 대한 통계분석을 수행하고, 각 유형별 평면에서 같은 유형의 잠재표현 노드가 활성화되는 영역을 BAM으로 시각화하였다. 분석 결과, 시기별로는 여러 유형이 연속적으로 서로 중첩되어 나타났다. 그 외의 특성에 대해서는, 한 개 유형이 원룸형 도시형생활주택만을 포함한 것을 제외하면, 특정 특성에서 배타적으로 나타나는 유형은 없었다. 각 유형별 BAM 시각화에 대한 분석을 통하여, 각 유형의 잠재표현 노드가 활성화되는 영역과 가장 활성화되는 평면의 주거공간 배치 특성을 도출하였다.

6장에서는 5장에서 도출된 주거공간 배치유형과 그 특성을 바탕으로 한국 아파트 단위평면의 진화과정을 해석하고, 그 결과를 선행연구와 비교하였다. 한국 아파트에서 전형적으로 나타나는 단위세대 규모 및 주동 형식 등에 따른 다양한 단위평면에 대하여, 해당 시기 배치유형에 대한 BAM 시각화 결과를 해석하였다. 1990년대 판상형 주동의 계단실형 단위평면에서 나타나는 다양한 주거공간 배치 특성을 잘 학습하였다. 특히, 2000년대 발코니 폭의 변화를 준공 시기와 단위평면만으로 도출한 것은 귀납적 분석의 가능성을 보여준다. 그러나 복도형 평면이나 탑상형 주동의 평면 등 의미는 크지 않지만 명확하게 구분할 수 있는 차이점이 있는 경우에는 그러한 특징을 제외한 다른 주거공간 배치는 무시되는 경향을 보였다. 또한, 소형 평면에서는 평면 외부의 빈 공간이 더 활성화되어 내부 주거공간 배치 분석에 영향을 주었다.

한국에서 아파트가 1천만 호가 넘게 공급된 시점에서 아파트에 대한 전수조

사는 곳 전국 주택에 대한 전수조사를 의미하게 되었다. 한 단지 안의 모든 단위세대가 손에 꼽을 수 있는 평면 유형으로 압축될 수 있지만, 그렇더라도 신축 주택의 상당수가 아파트인 상황에서는 파편화된 일부분을 따라가기에도 벅찰 수 밖에 없는 상황이었다. 개별 사례에 대한 연구자의 수작업이 필요하지 않은 방법론을 개발하여 한국 아파트에 대한 전수조사를 가능하게 한 점에 이 연구의 가장 중요한 의의가 있다.

또한, 준공 시기만 주고 학습한 딥 러닝 모형이 한국 아파트 사례에서 단위 평면의 주거공간 배치 특성을 도출할 수 있음을 보였다. 딥 러닝 모형의 설계 목적인 사진 속 사물 구분만큼 분류 기준과 분석 대상 사이에 직접적인 연관이 없는 과제에서도 준공 시기에 따른 주거공간 배치 계획 경향을 잘 찾아낼 수 있었다. 한국 아파트에 대한 건축 공간 연구에서 적용될 수 있는 가장 추상적인 기준에 따른 분석도 이 연구의 방법론을 적용하여 수행할 수 있음을 보임으로써 귀납적 건축 공간 연구의 가능성은 충분히 확인하였다.

이 연구의 방법론을 적용하여 분석한 한국 아파트 단위평면의 진화과정은 시기 별로 단위평면의 변화를 이끄는 원동력을 보여주었다. 이 연구에서도 복도형 및 계단실형 동선 계획의 차이, 발코니 면적 극대화를 위한 전면 폭 및 베이 수 증가, 탑상형 주동의 등장 및 혼합형 주동으로의 절충화, 한국 아파트에서 원룸형 규모의 확대 등의 계획 경향을 발견할 수 있었다. 이러한 결과는 선행연구의 발견과 같았으나, 그러나 그러한 결과가 도출되는 과정에서는 많은 차이가 있었다. 특히 시기별 단위평면을 유형화하는 선행연구에서는, 분석에 선행하여 시기 구분을 설정하는 경우가 대부분이었다. 이러한 연역적인 시기 설정은 각 시기를 단속적인 관계로 전제하게 된다. 따라서 서로 다른 시기에 속한 유사한 단위평면을 같은 유형으로 묶지 못하는 경우가 많이 나타났다. 반대로 이 연구에서 도출된 주거공간 배치유형은 최대 20년까지도 연속되는 유형이 서로 기간이 겹친 형태로 도출되었다. 이렇게 연역적인 연구 방법론의 제약을 뛰어넘어 귀납적인 접근을 가능하게 하는 것이 이 연구에서 개발한

분석 방법론의 의의이다.

이 연구의 방법론은 개별 사례에 대한 수작업이 필요하지 않고, 분석의 전체 과정이 재현가능하다는 점도 장점이다. 모든 자료에 대한 전처리와 딥 러닝 분석, 잠재표현 시각화 과정은 모두 명시적인 절차를 거쳐 수행되었을 뿐 아니라, 분석 과정 전체를 자동화하였다. 또한 누구나 분석에 사용된 코드를 사용하여 같은 분석을 재현하거나 분석 과정을 원하는 대로 변경하여 다시 수행할 수도 있다. 최근 재현가능한 연구에 대한 관심이 높아지고 있는 점을 고려하면 건축 공간 연구에서도 이러한 접근이 가능하다는 한 사례로서의 가치가 있다.

한편, 국민주택 규모에 맞춘 분석 영역을 적용하였기 때문에, 대형 평면 및 소형 평면의 분석에는 한계가 있었다. 단위평면의 축척을 동일하게 맞추면서 동시에 다양한 규모의 평면을 분석하는 것은 고정된 크기의 분석 영역을 입력받는 딥 러닝 모형으로는 어려움이 있다는 것을 확인하였다. 주거공간 배치 분석에 있어서 축척의 중요성은 다시 한번 검토가 필요한 부분이다. 그 결과에 따라 단위평면을 같은 크기로 맞추어 입력하거나, 분석 영역의 크기를 조절할 수 있는 방법을 탐색하는 등의 보완이 필요하다.

이 연구에서 구축한 한국 아파트 단위평면 데이터셋은 이 연구의 방법론과 함께 활용하여 향후 한국 아파트 단위평면에 대한 다양한 연구에 활용할 수 있는 결과물이다. 공개된 비정형 데이터를 기반으로 이러한 데이터셋을 구축하여, 앞으로 건축 빅데이터를 활용한 연구의 기반으로서의 의의가 있다. 현재는 한 출처에서 공개된 자료로만 구축되어 여러 한계가 있으나, 추가적인 공개 자료 수집과 함께 향후 세움터 등 공공 데이터를 활용하여 데이터셋을 확장할 수 있을 것으로 기대한다.

또한, 딥 러닝 방법론과 단위평면 데이터셋은 범주별 분류 및 유형화 연구 외에도 다양한 분야에 적용할 수 있는 도구이다. 2장에서 살펴본 바와 같이, 딥 러닝은 한국 아파트 단위평면 데이터셋에 대한 학습을 통하여 새로운 단위

평면의 생성이나 설계 보조 도구로서 설계자가 의도한 평면의 자동완성 등에 활용될 수 있다.

위와 같은 검토를 바탕으로, 추후 연구 과제는 크게 세 가지 방향으로 제시할 수 있다. 하나는 한국 아파트의 주거공간 배치를 다양한 기준에 따라 분석하는 연구이다. 이 연구에서는 방법론 검증을겸하여 딥 러닝 모형에 시기 구분을 학습시키고 시간에 따른 진화과정을 분석하였다. 그러나 이 연구에서 개발한 방법론과 한국 아파트 데이터셋을 활용하여, 주동 형식, 동선 계획, 발코니 확장 전제 등 주거공간 배치와 더 직접적인 연관이 있는 특성에 대한 분석도 가능하다. 이 연구의 방법론을 적용하여 한국 아파트에 대한 다양한 주제의 연구를 전수조사로 수행할 수 있다.

다음으로, 이 연구에서 제안한 귀납적 건축 공간 분석 방법론을 한국 아파트 뿐만 아니라 다른 건축 연구로 확장하는 것이다. 딥 러닝을 활용한 주거공간 배치 분석 방법론은 본질적으로 한국 아파트나 단위평면에 한정된 것이 아니다. 이 연구의 방법론을 단독주택부터 한옥까지 다양한 대상에 적용하고, 건축적 특성부터 사회문화적 요인까지 다양한 분석을 수행하는 것 또한 앞으로 나아갈 방향이다.

마지막으로, 한국 아파트 단위평면 데이터셋을 활용하여 딥 러닝을 적용한 단위평면의 생성적 설계 (generative design)를 위한 방법론을 개발하는 것은 한국 아파트에 딥 러닝을 적용하는 연구의 최종적 목표가 될 것이다. 물론, 딥 러닝을 포함한 기계학습 방법론은 과거의 설계안에서만 학습할 수 있기 때문에, 한국 아파트가 앞으로 대면할 미래의 새로운 문제를 해결하는 것은 건축가의 몫으로 남을 것이다. 그러나 한국 아파트가 한국에 토착화하고 또 새로운 변화를 이루어온 과정은 점진적인 개선의 누적이었다. 이를 뒷받침하는 데 생성적 설계의 역할과 과제가 있다.

# 참고문헌

## 단행본

Chaillou, S. (2019). AI + Architecture (mathesis). Harvard University. <https://www.academia.edu/39599650/>에서 참고

Gelézeau, V. (2004). 한국의 아파트연구 : 서울지역 7개 아파트단지의 경관분석을 중심으로. 아연.

Gelézeau, V. (2007). 아파트 공화국 : 프랑스 지리학자가 본 한국의 아파트. 후마니타스.

Goodfellow, I., Bengio, Y., Courville, A. (2016). Deep Learning. MIT Press.

Hanson, J. (2003). Decoding homes and houses. Cambridge university press.

Hillier, B., Hanson, J. (1989). The social logic of space. Cambridge university press.

공동주택연구회. (1999). 한국 공동주택계획의 역사. 世進社.

국토교통부. (2016). 2016년도 주거실태조사 통계보고서. <http://stat.molit.go.kr/portal/cate/statMetaView.do?hRsId=327>에서 참고

국토연구원. (2005). 현대 공간이론의 사상가들. 한울.

국토주택정보처. (2009). 토지주택핸드북. 토지주택핸드북. 한국토지주택공사 국토주택정보처.

대한주택공사. (2002). 주택도시 40년. 대한주택공사.

- 박용환. (2010). 한국근대주거론. 기문당.
- 박인석. (2013). 아파트 한국사회 : 단지 공화국에 간한 도시와 일상. 현암사.
- 박철수. (2006). 아파트의 문화사. 살림.
- 박해천. (2011). 콘크리트 유토피아. 자음과모음.
- 박해천. (2013). 아파트 게임 : 그들이 중산층이 될 수 있었던 이유. 휴머니스트.
- 손세관. (2016). 이십세기 집합주택 : 근대 공동주거 백 년의 역사. 열화당.
- 이진경. (2007). 근대적 주거공간의 탄생 (개정판..). 그린비.
- 장립종. (2009). 대한민국 아파트 발굴사 : 종암에서 힐탑까지, 1세대 아파트 탑사의 기록. 효형.
- 전남일. (2009). 한국 주거의 미시사. 돌베개.
- 전남일. (2010). 한국 주거의 공간사. 돌베개.
- 전남일. (2015). 집 : 집의 공간과 풍경은 어떻게 달라져 왔을까. 돌베개.
- 전남일, 손세관, 양세화, 홍형옥. (2008). 한국 주거의 사회사. 돌베개.
- 전봉희, 권용찬. (2012). 한옥과 한국 주택의 역사. 동녘.
- 전상인. (2009). 아파트에 미치다 : 현대한국의 주거사회학. 디자인하우스.
- 최두호. (2010). 아파트를 새롭게 디자인하라. 건축도시공간연구소 : auri.
- 최윤경. (2003). (7개의 키워드로 읽는)사회와 건축공간. 시공문화사.

## 학술논문

- Byun, N., Choi, J. (2016). A Typology of Korean Housing Units: In Search of Spatial Configuration. Journal of Asian Architecture and Building Engineering, 15(1), 41-48.

Elgammal, A., Mazzone, M., Liu, B., Kim, D., Elhoseiny, M. (2018). The Shape of Art History in the Eyes of the Machine. <http://arxiv.org/abs/1801.07729> 에서 참고

Eykholt, K., Evtimov, I., Fernandes, E., Li, B., Rahmati, A., Xiao, C., … Song, D. (2017). Robust Physical-World Attacks on Deep Learning Models. <http://arxiv.org/abs/1707.08945> 에서 참고

Huang, W., Zheng, H. (2018). Architectural drawings recognition and generation through machine learning.

Isola, P., Zhu, J.-Y., Zhou, T., Efros, A. A. (2016). Image-to-Image Translation with Conditional Adversarial Networks. arxiv.

Merrell, P., Schkufza, E., Koltun, V. (2010). Computer-generated residential building layouts. In ACM SIGGRAPH Asia 2010 papers (pp 1–12).

Newton, D. (2018). Multi-objective qualitative optimization (MOQO) in architectural design. In Proceedings of the 36th International Conference on Education and Research in Computer Aided Architectural Design in Europe, Poland.

Newton, D. (2019). Generative Deep Learning in Architectural Design. Technology|Architecture+ Design, 3(2), 176–189.

Rodrigues, E., Gaspar, A. R., Gomes, Á. (2013). An approach to the multi-level space allocation problem in architecture using a hybrid evolutionary technique. Automation in Construction, 35, 482–498.

Rodrigues, E., Sousa-Rodrigues, D., Sampayo, M. T. de, Gaspar, A. R., Gomes, Á., Antunes, C. H. (2017). Clustering of architectural floor plans: A comparison of shape representations. Automation in Construction, 80, 48–65.

Seo, K. W. (2003). Topological paths in housing evolution. In Proceedings of 4th Space Syntax International Symposium. London. <http://www.spacesyntax.net/symposia-archive/SSS4/fullpapers/80Seopaper.pdf> 에서 참고

Seo, K. W. (2007). Space puzzle in a concrete box: finding design competence that

generates the modern apartment houses in Seoul. Environment and Planning B: Planning and Design, 34(6), 1071–1084.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Driessche, G. van den, … Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. Nature, 529(7587), 484–489. <https://doi.org/10.1038/nature16961>

Simonyan, K., Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. <http://arxiv.org/abs/1409.1556> 에서 참고

Turner, A., Doxa, M., O’Sullivan, D., Penn, A. (2001). From Isovists to Visibility Graphs: A Methodology for the Analysis of Architectural Space. Environment and Planning B: Planning and Design, 28(1), 103–121. <https://doi.org/10.1068/b2684>

Yoshimura, Y., Cai, B., Wang, Z., Ratti, C. (2019). Deep learning architect: classification for architectural design through the eye of artificial intelligence. In International Conference on Computers in Urban Planning and Urban Management (pp 249–265). Springer.

Zeiler, M. D., Fergus, R. (2014). Visualizing and understanding convolutional networks. In European conference on computer vision (pp 818–833). Springer.

Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., Torralba, A. (2015). Learning Deep Features for Discriminative Localization. <http://arxiv.org/abs/1512.04150> 에서 참고

권혁삼, 전우현, 박준영. (2006). 공동주택 발코니 공간의 효율적 활용방안 연구. 대한건축학회 학술발표대회 논문집-계획계, 26(1), 81–84.

김성규, 김경배. (2016). 한국과 중국의 아파트 평면계획 특성과 주생활 문화의 관계 연구. 한국도시설계학회지 도시설계, 17(5), 49–61.

김수암, 김상호. (1997). 우리나라 민간아파트 주호평면의 시계열적 흐름에 관한 연구. 주택연구, 9, 103–127.

단경위, 신경주. (2012). 한국과 중국 아파트의 평면구성 비교. 한국실내디자인학회논문집, 21(5), 46–56.

도연정, 전봉희. (2017). 한국 아파트 평면에서 LDK 형성과정의 특성: 1962년-1988년 대한주택공사 아파트 사례를 중심으로. *대한건축학회 논문집-계획계*, 33(9), 61-70.

리광철, 이상현. (2008). 공간구문론을 이용한 한국과 중국의 아파트평면 공간구성방식에 관한 비교연구. *대한건축학회 논문집-계획계*, 24(9), 37-46.

박인석, 박노학, 천현숙. (2014). 전용면적 산정기준 변화와 발코니 용도변환 허용이 아파트 단위주거 평면설계에 미친 영향: 전용면적 60m<sup>2</sup> 와 85m<sup>2</sup> 평면의 실별 규모 변화를 중심으로. *한국주거학회논문집*, 25(2), 27-36.

배상영, 이재원, 이상엽. (2018). 공동주택 평면특성의 가격영향에 관한 연구 - 강남3구의 2005년 이후 분양주택을 중심으로 -. *한국건설관리학회논문집*, 19(4), 102-110. <https://doi.org/https://doi.org/10.6106/KJCEM.2018.19.4.102>

배연희, 하미경. (2019). 최근 공동주택의 주동형태 및 단위세대 평면 유형에 관한 연구-2019년 살기 좋은 아파트 를 중심으로. *한국실내디자인학회 논문집*, 28(6), 86-95.

신중진, 서기영, 허지연, 김홍룡, 김창수. (2002). 최근 초고층 아파트의 단위세대 평면계획특성에 관한 연구 (A Study on the Unit Plan Characteristics of the Recent Super-high-rise-Apartment). *대한건축학회 논문집-계획계*, 18(8), 11-22.

윤효진. (2019). 아파트의 주동형태 및 확장형 발코니에 따른 단위세대 평면계획 변화특성. *한국퍼실리티매니지먼트학회지*, 14(1), 61-69.

이병호, 이건원, 여영호. (2010). 공동주택 단위주호의 공간구성유형에 따른 정량적 평가지표에 관한 연구. *한국생태환경건축학회 논문집*, 10(5), 43-55.

이상진, 박소현. (2019). 부산 아파트 단지의 주동 평면형태의 변화 특성에 관한 연구. *대한건축학회 논문집-계획계*, 35(8), 3-14.

이지은, 이강업. (2014). 아파트의 평면 형태에 따른 채광 및 에너지성능 비교. *대한건축학회 논문집-계획계*, 30(3), 113-120.

전상인. (2007). 아파트 선호의 문화사회학. *환경논총*, 45, 11-32. <http://www.dbpia.co.kr/Article/NODE01793395> 에서 참고

정창용, 강부성, 김성규, 김진욱. (2008). 한국과 인도의 공동주택 단위평면 비교연구. *대한건*

- 축학회 논문집-계획계, 24(4), 45–52.
- 조극래, 박몽섭. (2006). 외기인접방식별 공동주택 단위평면 특성분석에 관한 연구. 대한건축학회 논문집-계획계, 22(1), 67–74.
- 천현숙. (2003). 아파트 주거 확산 요인에 관한 연구. 국토연구, 37, 65–81.
- 최권종, 진정. (2015). 국민주택 (전용 85m<sup>2</sup> 이하) 아파트평면의 변화에 대한 연구: 제도적 변화와 사회적 변화를 중심으로. 한국주거학회논문집, 26(5), 123–131.
- 최병숙, 박정아. (2009). 여성관련 공간을 중심으로 본 서울지역 아파트의 공간구조 변화. 한국주거학회논문집, 20(4), 39–47.
- 최재필. (1990). 중정에서 거실로-아파트 대량생산에 있어서의 거주성 확보를 위한 시험적 고찰. 주택도시, 51, 52–64.
- 최재필. (1996). 공간구문론을 사용한 국내 아파트 단위주호 평면의 시계열적 분석-수도권 4LDK 아파트 단위주호 평면계획을 중심으로. 대한건축학회 논문집, 12(7), 15–27.
- 최재필. (1996). 한국 현대 사회의 주생활양식의 변화-수도권 3LDK 아파트 주호 평면 계획의 변천을 중심으로. 대한건축학회 논문집, 12(9), 3–12.
- 최재필, 신재섭, 최영준. (2013). 국내 초고층 주거건축 평면구성 원리의 변천에 관한 연구. 대한건축학회 논문집-계획계, 29(2), 123–130.
- 최재필, 조형규, 박인수, 박영섭. (2004). 국내 아파트 단위주호 평면의 공간 분석-1966 년 2002 년의 서울지역 아파트를 대상으로. 대한건축학회 논문집-계획계, 20(6), 153–162.
- 최재필, 최준호, 박찬영. (2016). 리모델링을 대비한 1 기 신도시 노후 공동주택의 대표 유형에 관한 연구. 대한건축학회 논문집-계획계, 32(4), 33–40.
- 황용하, 최재필. (2003). 시각적 접근·노출 모델의 재고찰. 대한건축학회 논문집-계획계, 19(3), 11–18.

## 소프트웨어

Bradski, G. (2000). The OpenCV Library. Dr. Dobb's Journal of Software Tools.

Dhillon, I. S. (2001). Co-clustering documents and words using bipartite spectral graph partitioning. In Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining (pp 269–274).

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825–2830.

Rosebrock, A. (2018). Keras and Convolutional Neural Networks (CNNs). <https://www.pyimagesearch.com/2018/04/16/keras-and-convolutional-neural-networks-cnns/> 에서 참고

scikit-learn developers. (2019). Classifier comparison. [https://scikit-learn.org/stable/auto\\_examples/classification/plot\\_classifier\\_comparison.html](https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html) 에서 참고

scikit-learn developers. (2020). A demo of the Spectral Co-Clustering algorithm. [https://scikit-learn.org/stable/auto\\_examples/bicluster/plot\\_spectral\\_coclustering.html](https://scikit-learn.org/stable/auto_examples/bicluster/plot_spectral_coclustering.html) 에서 참고

scikit-learn developers. (2020). Biclustering. <https://scikit-learn.org/stable/modules/biclustering.html> 에서 참고

Sculley, D. (2010). Web-scale k-means clustering. In Proceedings of the 19th international conference on World wide web (pp 1177–1178).

## 기타

Anderson, C. (2008). The end of theory: The data deluge makes the scientific method obsolete. Wired magazine. <https://www.wired.com/2008/06/pb-theory/> 에서 참고

NAVER Corp. (2017). 네이버 부동산. <https://land.naver.com/> 에서 참고

Piatetsky-Shapiro, G. (2016). Data science Venn diagram. <https://www.kdnuggets.com/2016/07/data-science-venn-diagram.html>

com/2016/03/data-science-puzzle-explained.html 에서 참고

Project Open Data. (2014). Project Open Data Metadata Schema v1.1. <https://project-open-data.cio.gov/v1.1/schema/> 에서 참고

이제훈, 이종규. (1999). 마침내 ‘아파트 공화국’. 한겨레신문, 17. <https://newslibrary.naver.com/viewer/index.nhn?articleId=1999081000289117001> 에서 참고

통계청. (2018). 주택의 종류별 주택-읍면동(2015), 시군구(2016~). [http://kosis.kr/statHtml/statHtml.do?orgId=101&tblId=DT\\_1JU1501](http://kosis.kr/statHtml/statHtml.do?orgId=101&tblId=DT_1JU1501) 에서 참고

# 부록

## Source code

이 연구에서 작성된 소스코드는 GitHub 저장소( <https://github.com/esnahn/NaverFloorplanAnalysis> )를 통해서도 공개하였다.

### 10\_naver\_floorplan\_analysis.py

```
1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # In[1]:
5
6
7 get_ipython().run_line_magic("matplotlib", "inline")
8
9 import cv2, matplotlib
10 import numpy as np
11 from skimage.morphology import (
12     skeletonize,
13     skeletonize_3d,
14     medial_axis,
15     thin,
16     local_minima,
17     local_maxima,
18 )
19 from scipy.ndimage import distance_transform_edt
20
21 from math import sqrt
22
```

```

23 import matplotlib.pyplot as plt
24
25 from os.path import expanduser, splitext
26 from os import scandir, makedirs
27
28 # import random
29
30 import csv
31
32 from tqdm import tnrange, tqdm_notebook
33
34 from pathlib import Path
35
36 debug = True # plot every steps
37
38
39 # In[2]:
40
41
42 def read_from_csv(filepath):
43     if Path(filepath).is_file():
44
45         with open(filepath, "r", newline="", encoding="utf-8-sig") as csvfile:
46             listreader = csv.reader(csvfile)
47             columns = next(listreader)
48             readlist = list(listreader)
49
50     else:
51         columns = []
52         readlist = []
53
54     return columns, readlist
55
56
57 def read_bgr_from_image_unicode(path):
58     """workaround for non-ascii filenames"""
59
60     stream = open(path, "rb")
61     bytes_ = bytearray(stream.read())
62     numpyarray = np.asarray(bytes_, dtype=np.uint8)
63     bgr = cv2.imdecode(numpyarray, cv2.IMREAD_UNCHANGED)
64
65     return bgr
66

```

```

67
68 def save_bgr_to_image_unicode(bgr, path, ext_to=".png"):
69     """workaround for non-ascii filenames"""
70
71     _, numpyarray = cv2.imencode(ext_to, bgr)
72     with open(path, "wb") as file:
73         file.write(numpyarray)
74
75
76 # # unit mask
77
78 # In[3]:
79
80
81 def color_dict_mask(
82     img_dict={
83         "Lab": np.zeros((1, 1, 3), dtype="uint8"),
84         "HSV": np.zeros((1, 1, 3), dtype="uint8"),
85     },
86     colors={
87         "colorname": {
88             "Lab": ([0, 0, 0], [255, 255, 255]),
89             "HSV": ([0, 0, 0], [255, 255, 255]),
90         }
91     },
92 ):
93     # get masks matching any of the colors matching all descriptions
94
95     mask = np.zeros_like(list(img_dict.values())[0][:, :, 0])
96     for color_dict in colors.values():
97         mask_color = np.ones_like(mask) * 255
98         for colorspace, limits in color_dict.items():
99             mask_colorspace = cv2.inRange(
100                 img_dict[colorspace], np.array(limits[0]), np.array(limits[1]))
101
102             mask_color = cv2.bitwise_and(mask_color, mask_colorspace)
103
104     mask = cv2.bitwise_or(mask, mask_color)
105
106     return mask
107
108
109 def get_color_mask(
110     blur={
```

```

111         "Lab": np.zeros((1, 1, 3), dtype="uint8"),
112         "HSV": np.zeros((1, 1, 3), dtype="uint8"),
113     },
114     colors={
115         "colorname": {
116             "Lab": ([0, 0, 0], [255, 255, 255]),
117             "HSV": ([0, 0, 0], [255, 255, 255]),
118         }
119     },
120 ):
121     #     lab = cv2.cvtColor(bgr, cv2.COLOR_BGR2Lab)
122
123     #     blur = {}
124     #     blur["Lab"] = cv2.bilateralFilter(lab, 15, 25, 150)
125     #     blur["BGR"] = cv2.cvtColor(blur["Lab"], cv2.COLOR_Lab2BGR)
126     #     blur["HSV"] = cv2.cvtColor(blur["BGR"], cv2.COLOR_BGR2HSV)
127
128     # get masks matching any of the colors matching all descriptions
129
130     mask = color_dict_mask(blur, colors)
131
132     # fill holes and remove noise
133
134     contours, hierarchy = cv2.findContours(mask, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
135
136     holes = [contours[i] for i in range(len(contours)) if hierarchy[0][i][3] >= 0]
137     cv2.drawContours(mask, holes, -1, 255, -1)
138
139     kernel_5c = np.array(
140         [
141             [0, 1, 1, 1, 0],
142             [1, 1, 1, 1, 1],
143             [1, 1, 1, 1, 1],
144             [1, 1, 1, 1, 1],
145             [0, 1, 1, 1, 0],
146         ],
147         dtype=np.uint8,
148     )
149
150     kernel_9c = np.zeros((9, 9), np.uint8)
151     cv2.circle(kernel_9c, (4, 4), 4, 1, -1)
152
153     kernel_15c = np.zeros((15, 15), np.uint8)
154     cv2.circle(kernel_15c, (7, 7), 7, 1, -1)

```

```

155
156     # mask = cv2.erode(mask, kernel_5c, iterations=1)
157
158     smallbits = [
159         contours[i]
160         for i in range(len(contours))
161         if hierarchy[0][i][3] == -1 and cv2.contourArea(contours[i]) <= 100
162     ]
163     cv2.drawContours(mask, smallbits, -1, 0, -1)
164
165     # removing imperfections
166
167     contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
168
169     for c in contours:
170         if cv2.contourArea(c) >= 100:
171             mask_single_c = np.zeros_like(mask)
172             cv2.drawContours(mask_single_c, c, -1, 255, -1)
173
174             mask_single_c = cv2.morphologyEx(
175                 mask_single_c, cv2.MORPH_CLOSE, kernel_9c, iterations=1
176             )
177             mask |= mask_single_c
178
179     return mask
180
181
182 def get_marked_contours(contours, marker_mask, min_marked_area):
183     marked_contours = []
184
185     for c in contours:
186         mask_single_c = np.zeros_like(marker_mask)
187         cv2.drawContours(mask_single_c, [c], -1, 255, -1)
188
189         c_area = cv2.countNonZero(mask_single_c)
190         marked_area = cv2.countNonZero(mask_single_c & marker_mask)
191
192         if marked_area >= min_marked_area:
193             marked_contours.append(c)
194
195     return marked_contours
196
197
198 def get_marked_mask(boundary_mask, marker_mask, min_marked_area):

```

```

199     contours, hierarchy = cv2.findContours(
200         boundary_mask, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE
201     )
202
203     marked_contours = get_marked_contours(contours, marker_mask, min_marked_area)
204
205     marked_mask = np.zeros_like(boundary_mask)
206
207     if marked_contours:
208         cv2.drawContours(marked_mask, marked_contours, -1, 255, -1)
209
210     return marked_mask
211
212
213 def get_wall_mask(bgr=np.zeros((1, 1, 3), dtype="uint8")):
214
215     kernel_3 = np.ones((3, 3), np.uint8)
216     kernel_5c = np.array(
217         [
218             [0, 1, 1, 1, 0],
219             [1, 1, 1, 1, 1],
220             [1, 1, 1, 1, 1],
221             [1, 1, 1, 1, 1],
222             [0, 1, 1, 1, 0],
223         ],
224         dtype=np.uint8,
225     )
226
227     # get mask based on color and shape
228
229     redimg = bgr[:, :, 2]
230     _, threshold_img_inv = cv2.threshold(redimg, 140, 255, cv2.THRESH_BINARY_INV)
231     # plt.imshow(threshold_img_inv)
232
233     threshold.blur = cv2.medianBlur(threshold_img_inv, 5)
234     # plt.imshow(threshold.blur)
235     erosion = cv2.erode(threshold.blur, kernel_3)
236     opening = cv2.morphologyEx(threshold.blur, cv2.MORPH_OPEN, kernel_3)
237     # dilation = cv2.dilate(opening, kernel_3)
238     # plt.imshow(opening)
239     mask = cv2.bitwise_and(threshold_img_inv, opening)
240     # plt.figure()
241     # plt.imshow(mask)
242

```

```

243     kernel = kernel_5c
244
245     ret, markers = cv2.connectedComponents(mask)
246     #     plt.figure()
247     #     plt.imshow(markers)
248
249     wall_mask = np.zeros_like(mask)
250     for i in range(1, ret):
251         if (markers == i).sum() > 300:
252             wall_mask |= (markers == i).astype(np.uint8) * 255
253     #     plt.figure()
254     #     plt.imshow(wall_mask)
255
256     return wall_mask
257
258
259 def get_LDK_mask(
260     blur={
261         "Lab": np.zeros((1, 1, 3), dtype="uint8"),
262         "HSV": np.zeros((1, 1, 3), dtype="uint8"),
263     },
264 ):
265     floor_colors = {
266         "floor_light": {
267             "Lab": ([180, 130, 160], [220, 150, 190]),
268             "HSV": ([0, 65, 180], [20, 255, 255]),
269         },
270         "floor_dark": {
271             "Lab": ([120, 130, 150], [180, 155, 190]),
272             "HSV": ([0, 90, 100], [20, 255, 230]),
273         },
274         "floor_watermark": {
275             "Lab": ([220, 125, 145], [240, 145, 165]),
276             "HSV": ([0, 65, 220], [20, 255, 255]),
277         },
278     }
279
280     mask = get_color_mask(blur, floor_colors)
281
282     return mask
283
284
285 def get_bedroom_mask(
286     blur={
```

```

287         "Lab": np.zeros((1, 1, 3), dtype="uint8"),
288         "HSV": np.zeros((1, 1, 3), dtype="uint8"),
289     },
290 ): bedroom_boundary = {
291     "bedroom_boundary": {
292         "Lab": ([180, 120, 132], [254, 135, 165]),
293         "HSV": ([10, 25, 200], [30, 110, 255]),
294     }
295 }
296 bedroom_dark = {
297     "bedroom_dark": {
298         "Lab": ([160, 124, 139], [250, 130, 165]),
299         "HSV": ([10, 30, 200], [30, 90, 250]),
300     }
301 }
302 balcony_colors = {"balcony": {"Lab": ([240, 125, 130], [254, 135, 140])}}
303
304 bedroom_boundary_mask = get_color_mask(blur, bedroom_boundary)
305 bedroom_dark_mask = get_color_mask(blur, bedroom_dark)
306 balcony_mask = get_color_mask(blur, balcony_colors)
307
308 # remove balconies which is similarly colored
309
310 mask_bedroom_only = np.zeros_like(bedroom_boundary_mask)
311
312 contours, _ = cv2.findContours(
313     bedroom_boundary_mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE
314 )
315
316 for c in contours:
317     mask_single_c = np.zeros_like(mask_bedroom_only)
318     cv2.drawContours(mask_single_c, [c], -1, 255, -1)
319
320     c_area = cv2.countNonZero(mask_single_c)
321     dark_area = cv2.countNonZero(mask_single_c & bedroom_dark_mask)
322     balcony_area = cv2.countNonZero(mask_single_c & balcony_mask)
323
324     if dark_area >= 1000:
325         mask_bedroom_only |= mask_single_c
326
327 return mask_bedroom_only
328
329
330 def get_balcony_mask(

```

```

331     blur={
332         "Lab": np.zeros((1, 1, 3), dtype="uint8"),
333         "HSV": np.zeros((1, 1, 3), dtype="uint8"),
334     },
335 ):
336     balcony_boundary = {
337         "bedroom_boundary": {
338             "Lab": ([180, 120, 132], [254, 135, 165]),
339             "HSV": ([10, 15, 200], [30, 110, 255]),
340         }
341     }
342     bedroom_dark = {
343         "bedroom_dark": {
344             "Lab": ([160, 124, 139], [250, 130, 165]),
345             "HSV": ([10, 30, 200], [30, 90, 250]),
346         }
347     }
348     balcony_colors = {"balcony": {"Lab": ([240, 125, 130], [254, 135, 140])}}
349
350     balcony_boundary_mask = get_color_mask(blur, balcony_boundary)
351     bedroom_dark_mask = get_color_mask(blur, bedroom_dark)
352     balcony_mask = get_color_mask(blur, balcony_colors)
353
354     # remain balconies only
355
356     mask_balcony_only = np.zeros_like(balcony_boundary_mask)
357
358     contours, _ = cv2.findContours(
359         balcony_boundary_mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE
360     )
361
362     for c in contours:
363         mask_single_c = np.zeros_like(mask_balcony_only)
364         cv2.drawContours(mask_single_c, [c], -1, 255, -1)
365
366         c_area = cv2.countNonZero(mask_single_c)
367         dark_area = cv2.countNonZero(mask_single_c & bedroom_dark_mask)
368         balcony_area = cv2.countNonZero(mask_single_c & balcony_mask)
369
370         if dark_area <= balcony_area and 10 <= balcony_area:
371             mask_balcony_only |= mask_single_c
372
373     return mask_balcony_only
374

```

```

375 def get_entrance_mask(bgr=np.zeros((1, 1, 3), dtype="uint8")):
376     entrance_boundary = {"white_and_gray": {"HSV": ([0, 0, 170], [255, 20, 255])}}
377     white = {"white": {"HSV": ([0, 0, 245], [255, 10, 255])}}
378     gray = {"gray": {"HSV": ([0, 0, 230], [255, 10, 245])}}
379
380     lab = cv2.cvtColor(bgr, cv2.COLOR_BGR2Lab)
381
382     blur = {}
383     blur["Lab"] = cv2.bilateralFilter(lab, 15, 5, 150)
384     blur["BGR"] = cv2.cvtColor(blur["Lab"], cv2.COLOR_Lab2BGR)
385     blur["HSV"] = cv2.cvtColor(blur["BGR"], cv2.COLOR_BGR2HSV)
386
387     kernel_3 = np.ones((3, 3), np.uint8)
388     kernel_5c = np.array(
389         [
390             [0, 1, 1, 1, 0],
391             [1, 1, 1, 1, 1],
392             [1, 1, 1, 1, 1],
393             [1, 1, 1, 1, 1],
394             [0, 1, 1, 1, 0],
395         ],
396         dtype=np.uint8,
397     )
398     kernel_7c = np.zeros((7, 7), np.uint8)
399     cv2.circle(kernel_7c, (3, 3), 3, 1, -1)
400     kernel_9c = np.zeros((9, 9), np.uint8)
401     cv2.circle(kernel_9c, (4, 4), 4, 1, -1)
402     kernel_15c = np.zeros((15, 15), np.uint8)
403     cv2.circle(kernel_15c, (7, 7), 7, 1, -1)
404
405     mask_e, mask_w, mask_g = [
406         color_dict_mask(blur, x) for x in [entrance_boundary, white, gray]
407     ]
408     area_e, area_w, area_g = [cv2.countNonZero(x) for x in [mask_e, mask_w, mask_g]]
409
410     mask_e_e = cv2.erode(mask_e, kernel_7c)
411
412     mask_w_d, mask_g_d = [cv2.dilate(x, kernel_15c) for x in [mask_w, mask_g]]
413     mask_wg_c = cv2.erode(mask_w_d & mask_g_d, kernel_15c)
414
415     #     if debug:
416     #         print(area_e, area_w, area_g)
417     #         plt.figure()
418     #         plt.imshow(mask_e_e & 32 | mask_wg_c & 128, cmap="binary")

```

```

419
420     contours, hierarchy = cv2.findContours(
421         mask_e_e & mask_wg_c, cv2.RETR_CCOMP, cv2.CHAIN_APPROX_NONE
422     )
423
424     mask_ent = np.zeros_like(mask_e)
425
426     for i in range(len(contours)):
427         if hierarchy[0][i][3] == -1:
428             cnt = contours[i]
429             mask_c = np.zeros_like(mask_ent)
430             cv2.drawContours(mask_c, [cnt], -1, 255, -1)
431
432             area_c = cv2.countNonZero(mask_c & mask_e)
433             area_c_w = cv2.countNonZero(mask_c & mask_w)
434             area_c_g = cv2.countNonZero(mask_c & mask_g)
435
436             if (
437                 area_c >= 100
438                 and area_c >= 0.01 * area_g
439                 and area_c_w >= 0.3 * area_c
440                 and area_c_g >= 0.3 * area_c
441                 and area_c_w + area_c_g >= 0.8 * area_c
442             ):
443                 mask_ent |= mask_c
444
445     mask_ent = cv2.morphologyEx(mask_ent, cv2.MORPH_CLOSE, kernel_15c)
446
447     if debug:
448         fig = plt.figure(figsize=(3, 3), dpi=300)
449         plt.axes().axis("off")
450         plt.imshow(mask_ent & 128, cmap="binary")
451         plt.tight_layout()
452         #         fig.savefig("floorplan_entrance.pdf", bbox_inches="tight", pad_inches=0)
453
454     return mask_ent
455
456
457 def get_bathroom_mask(
458     blur={
459         "Lab": np.zeros((1, 1, 3), dtype="uint8"),
460         "HSV": np.zeros((1, 1, 3), dtype="uint8"),
461     },
462 ):

```

```

463     bathroom_colors = {"bathroom": {"HSV": ([90, 10, 220], [110, 40, 255])}}
464
465     mask = get_color_mask(blur, bathroom_colors)
466
467     return mask
468
469
470 def get_watershed(
471     thresh=np.zeros((1, 1), dtype="uint8"), markers=np.zeros((1, 1), dtype="uint8")
472 ):
473     unknown = cv2.subtract(thresh, markers.astype(thresh.dtype))
474
475     markers = markers.astype(np.int32)
476     markers = markers + 1
477     markers[unknown == 255] = 0
478
479     markers = cv2.watershed(np.stack([thresh] * 3, axis=2), markers)
480     markers = markers - 1
481     markers[markers <= 0] = 0
482
483     return markers
484
485
486 # In[4]:
487
488
489 # https://stackoverflow.com/questions/26537313/how-can-i-find-endpoints-of-binary-skeleton-image-in-opencv
490 def skeleton_endpoints(skel):
491     # make out input nice, possibly necessary
492     skel = skel.copy()
493     skel[skel != 0] = 1
494     skel = np.uint8(skel)
495
496     # apply the convolution
497     kernel = np.uint8([[1, 1, 1], [1, 10, 1], [1, 1, 1]])
498     src_depth = -1
499     filtered = cv2.filter2D(skel, src_depth, kernel)
500
501     # now look through to find the value of 11
502     # this returns a mask of the endpoints, but if you just want the coordinates, you could simply return np.where(filtered==11)
503     out = np.zeros_like(skel)
504     out[np.where(filtered == 11)] = 1
505
506     return out

```

```

507
508 # In[5]:
509
510
511 def get_unit_mask(bgr=np.zeros((1, 1, 3), dtype="uint8")):
512     """Returns unit plan masks of the unit plan,
513     as a dictionary of opencv masks and also a single combined mask,
514     including masks for walls, entrances, LDK, bedrooms, balconies, and bathrooms."""
515
516     AREA_UNIT = 128
517     AREA_WALL = 64
518     AREA_ENTRANCE = 32
519     AREA_LDK = 16
520     AREA_BEDROOM = 8
521     AREA_BALCONY = 4
522     AREA_BATHROOM = 2
523
524     kernel_3 = np.ones((3, 3), np.uint8)
525     kernel_5c = np.array(
526         [
527             [0, 1, 1, 1, 0],
528             [1, 1, 1, 1, 1],
529             [1, 1, 1, 1, 1],
530             [1, 1, 1, 1, 1],
531             [0, 1, 1, 1, 0],
532         ],
533         dtype=np.uint8,
534     )
535     kernel_7c = np.zeros((7, 7), np.uint8)
536     cv2.circle(kernel_7c, (3, 3), 3, 1, -1)
537     kernel_9c = np.zeros((9, 9), np.uint8)
538     cv2.circle(kernel_9c, (4, 4), 4, 1, -1)
539     kernel_15c = np.zeros((15, 15), np.uint8)
540     cv2.circle(kernel_15c, (7, 7), 7, 1, -1)
541
542     kernel_cross = np.array([[0, 1, 0], [1, 1, 1], [0, 1, 0]], dtype=np.uint8)
543
544     rgb = cv2.cvtColor(bgr, cv2.COLOR_BGR2RGB)
545     lab = cv2.cvtColor(bgr, cv2.COLOR_BGR2Lab)
546     hsv = cv2.cvtColor(bgr, cv2.COLOR_BGR2HSV)
547     img = {"BGR": bgr, "RGB": rgb, "Lab": lab, "HSV": hsv}
548
549     if debug:
550         fig = plt.figure(figsize=(6, 4), dpi=300)

```

```

551         plt.axes().axis("off")
552         plt.imshow(rgb)
553         plt.tight_layout()
554
555     blur = {"Lab": cv2.bilateralFilter(lab, 15, 25, 150)}
556     blur["BGR"] = cv2.cvtColor(blur["Lab"], cv2.COLOR_Lab2BGR)
557     blur["RGB"] = cv2.cvtColor(blur["BGR"], cv2.COLOR_BGR2RGB)
558     blur["HSV"] = cv2.cvtColor(blur["BGR"], cv2.COLOR_BGR2HSV)
559
560     if debug:
561         fig = plt.figure(figsize=(6, 4), dpi=300)
562         plt.axes().axis("off")
563         plt.imshow(blur["RGB"])
564         plt.tight_layout()
565
566 #####
567 # Get wall/indoor/outdoor markers #
568 #####
569
570 ### get wall
571
572 wall_mask = get_wall_mask(bgr)
573 wall_mask_d = cv2.dilate(wall_mask, kernel_9c)
574
575 # entrance
576 ent_mask = get_entrance_mask(bgr)
577 ent_mask_d = cv2.dilate(ent_mask, kernel_9c)
578
579 ### outside of the largest foreground area as outdoor boundary
580
581 white_color = {"white": {"HSV": ([0, 0, 245], [180, 10, 255])}}
582 white_mask = color_dict_mask({"HSV": blur["HSV"]}, white_color)
583
584 ret, markers = cv2.connectedComponents(~white_mask)
585 max_i = max(range(1, ret), key=lambda i: (markers == i).sum())
586 #     print(max_i)
587 mask = (markers == max_i).astype(np.uint8) * 255
588 mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel_15c)
589 contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
590 cv2.drawContours(mask, contours, -1, 255, -1)
591
592 outdoor_mask = cv2.morphologyEx(~mask, cv2.MORPH_CLOSE, kernel_9c)
593 outdoor_mask_d = cv2.dilate(outdoor_mask, kernel_9c)
594
```

```

595     #     if debug:
596     #         fig = plt.figure(figsize=(6, 4), dpi=300)
597     #         plt.axes().axis("off")
598     #         plt.imshow(
599     #             outdoor_mask, cmap="binary",
600     #         )
601     #         plt.tight_layout()
602
603 ######
604 # Getting color based masks      #
605 ######
606
607 #     wall_mask
608 #     ent_mask
609
610 ldk_mask = get_LDK_mask(blur)
611 bed_mask = get_bedroom_mask(blur)
612 bal_mask = get_balcony_mask(blur)
613 bath_mask = get_bathroom_mask(blur)
614
615 indoor_mask = ent_mask | ldk_mask | bed_mask | bal_mask | bath_mask
616
617 ### get bounding box of indoor mask
618
619 x, y, w, h = cv2.boundingRect(indoor_mask)
620 indoor_bbox = cv2.rectangle(
621     np.zeros_like(indoor_mask), (x, y), (x + w, y + h), 255, -1
622 ).astype(np.uint8)
623
624 ### make outmost zones do not contain LDK marker outdoor
625
626 zones = ~outdoor_mask_d & ~wall_mask_d
627 zones = cv2.dilate(zones, kernel_9c)
628
629 ret, markers = cv2.connectedComponents(zones)
630 for i in range(1, ret):
631     marker = (markers == i).astype(np.uint8) * 255
632     if not (marker & ldk_mask).sum() and (marker & outdoor_mask_d).sum():
633         outdoor_mask |= marker
634
635 ### regenerate masks
636
637 outdoor_mask = cv2.morphologyEx(outdoor_mask, cv2.MORPH_CLOSE, kernel_9c)
638 outdoor_mask_d = cv2.dilate(outdoor_mask, kernel_9c)

```

```

639
640     if debug:
641         fig = plt.figure(figsize=(6, 4), dpi=300)
642         plt.axes().axis("off")
643         plt.imshow(outdoor_mask, cmap="binary")
644         plt.tight_layout()
645
646 ######
647 # Skeleton of walls and space      #
648 ######
649
650 zones = ~wall_mask_d
651 #    zones = cv2.dilate(zones, kernel_9c)
652
653 skeleton, dist = medial_axis(zones, return_distance=True)
654 skeleton = skeleton.astype(np.uint8) * 255
655 ret, markers = cv2.connectedComponents(skeleton)
656
657 skel_indoor = np.zeros_like(skeleton)
658 for i in range(1, ret):
659     marker = (markers == i).astype(np.uint8) * 255
660     if cv2.countNonZero(marker & indoor_mask):
661         skel_indoor |= marker
662
663 if debug:
664     fig = plt.figure(figsize=(6, 4), dpi=300)
665     plt.axes().axis("off")
666     plt.imshow(skel_indoor | (wall_mask & 32), cmap="binary")
667     plt.tight_layout()
668
669 #####
670 # Get non-wall borders      #
671 #####
672
673 border = cv2.Canny(blur["RGB"], 100, 200) & ~ent_mask_d
674
675 if debug:
676     fig = plt.figure(figsize=(6, 4), dpi=300)
677     plt.axes().axis("off")
678     plt.imshow(border, cmap="binary")
679     plt.tight_layout()
680
681 ### pick borders touching walls and the skeleton
682

```

```

683     ret, markers = cv2.connectedComponents(border)
684     for i in range(1, ret):
685         marker = (markers == i).astype(np.uint8) * 255
686         if not ((marker & wall_mask).sum() and (marker & skel_indoor).sum()):
687             border &= ~marker
688
689     if debug:
690         fig = plt.figure(figsize=(6, 4), dpi=300)
691         plt.axes().axis("off")
692         plt.imshow(border | wall_mask_d & 32, cmap="binary")
693         plt.tight_layout()
694
695     ### if a white/gray space is larger than the smallest bedroom, it's outside
696
697     #     # size of the smallest bedroom (for determine a core)
698     #     min_bed_size = cv2.countNonZero(bed_mask)
699     #     ret, markers = cv2.connectedComponents(
700     #         cv2.morphologyEx(bed_mask, cv2.MORPH_CLOSE, kernel_9c) & ~wall_mask
701     #     )
702     #     for i in range(1, ret):
703     #         marker = (markers == i).astype(np.uint8) * 255
704     #         if cv2.countNonZero(marker) < min_bed_size:
705     #             min_bed_size = cv2.countNonZero(marker)
706     #     if debug:
707     #         print(min_bed_size)
708
709     zones = ~wall_mask & ~border
710     zones = cv2.morphologyEx(zones, cv2.MORPH_OPEN, kernel_5c)
711     ret, markers = cv2.connectedComponents(zones, connectivity=4)
712     if debug:
713         fig = plt.figure(figsize=(6, 4), dpi=300)
714         plt.axes().axis("off")
715         plt.imshow(markers, cmap="gist_ncar")
716         plt.tight_layout()
717
718         fig = plt.figure(figsize=(6, 4), dpi=300)
719         plt.axes().axis("off")
720         plt.imshow(markers % 20, cmap="tab20")
721         plt.tight_layout()
722
723     indoor_mask_area = cv2.countNonZero(indoor_mask)
724     for i in range(1, ret):
725         marker = (markers == i).astype(np.uint8) * 255
726         if not (marker & indoor_mask).sum():

```

```

727         if cv2.countNonZero(marker) > 0.10 * indoor_mask_area:
728             outdoor_mask |= marker
729
730     if debug:
731         fig = plt.figure(figsize=(6, 4), dpi=300)
732         plt.axes().axis("off")
733         plt.imshow(outdoor_mask | wall_mask & 32, cmap="binary")
734         plt.tight_layout()
735
736     ### add boundaries of color masks if a zone contains more than one color
737
738     del outdoor_mask_d
739
740     color_stacked = np.dstack(
741         (outdoor_mask, ent_mask_d, ldk_mask, bed_mask, bal_mask, bath_mask)
742     )
743     if debug:
744         print(color_stacked.shape)
745         print(
746             (
747                 np.expand_dims(zones > 0, axis=2) & cv2.dilate(color_stacked, kernel_9c)
748                 > 0
749             ).sum(axis=(0, 1))
750         )
751
752     edge_stacked = np.zeros_like(color_stacked)
753     for k in range(6):
754         edge_stacked[:, :, k] = cv2.Canny(color_stacked[:, :, k], 100, 200) & ~ent_mask
755     edge_combined = np.bitwise_or.reduce(edge_stacked, 2)
756
757     if debug:
758         fig = plt.figure(figsize=(6, 4), dpi=300)
759         plt.axes().axis("off")
760         plt.imshow(edge_combined, cmap="binary")
761         plt.tight_layout()
762
763     #     ret, markers = cv2.connectedComponents(zones, connectivity=4)
764     for i in range(1, ret):
765         marker = (markers == i).astype(np.uint8) * 255
766         indoor_areas = (np.expand_dims(marker > 0, axis=2) & color_stacked).sum(
767             axis=(0, 1)
768         )
769         if np.count_nonzero(indoor_areas) >= 2:
770             border |= marker & edge_combined

```

```

771
772     if debug:
773         fig = plt.figure(figsize=(6, 4), dpi=300)
774         plt.axes().axis("off")
775         plt.imshow(border, cmap="binary")
776         plt.tight_layout()
777
778 ######
779 # Fill zones           #
780 ######
781
782 wall_mask_3d = np.expand_dims(wall_mask, axis=2)
783 wall_mask_d_3d = np.expand_dims(wall_mask_d, axis=2)
784
785 color_stacked = (
786     np.dstack((outdoor_mask, ent_mask_d, ldk_mask, bed_mask, bal_mask, bath_mask))
787     & ~wall_mask_3d
788 )
789 zones_filled = np.zeros_like(color_stacked)
790
791 zones = ~wall_mask & ~border
792 zones = cv2.morphologyEx(zones, cv2.MORPH_OPEN, kernel_5c)
793
794 # remove area not touching indoor markers
795 ret, markers = cv2.connectedComponents(~wall_mask)
796 for i in range(1, ret):
797     marker = (markers == i).astype(np.uint8) * 255
798     if not ((marker & indoor_mask).sum()):
799         zones &= ~marker
800
801 if debug:
802     fig = plt.figure(figsize=(6, 4), dpi=300)
803     plt.axes().axis("off")
804     plt.imshow(zones, cmap="binary")
805     plt.tight_layout()
806
807 # make zones outside if more than a half of it is outside of bounding box (sanity check)
808
809 ret, markers = cv2.connectedComponents(zones, connectivity=4)
810 marker_stacked = np.dstack(
811     [(markers == i).astype(np.uint8) * 255 for i in range(ret)])
812 )
813 indexes = list(range(1, ret))
814

```

```

815     indoor_mask_area = cv2.countNonZero(indoor_mask)
816     margin = 0.02 * indoor_mask_area
817
818     for i in indexes:
819         marker = marker_stacked[:, :, i]
820         if cv2.countNonZero(marker) % 2 > (
821             cv2.countNonZero(marker & indoor_bbox) # + margin
822         ):
823             indexes.remove(i)
824             zones &= ~marker
825
826         # outdoor
827         color_stacked[:, :, 0] |= marker
828         zones_filled[:, :, 0] |= marker
829
830         # fill
831         count_last = len(indexes)
832         remove_indexes = []
833         repeat = 0
834         while indexes:
835             if debug:
836                 print(cv2.countNonZero(zones))
837
838             for i in indexes:
839                 marker = marker_stacked[:, :, i]
840                 indoor_areas = (np.expand_dims(marker > 0, axis=2) & color_stacked > 0).sum(
841                     axis=(0, 1)
842                 )
843                 k = indoor_areas.argmax()
844
845                 if debug:
846                     print(i, k, indoor_areas[k])
847
848                 if indoor_areas[k]:
849                     if k != 0 or indoor_areas[1]:
850                         remove_indexes.append(i)
851                         zones &= ~marker
852
853                     color_stacked[:, :, k] |= marker
854                     zones_filled[:, :, k] |= marker
855
856             indexes = [i for i in indexes if i not in remove_indexes]
857
858             if len(indexes) == count_last:

```

```

859         color_stacked = cv2.dilate(color_stacked, kernel_15c)
860         color_stacked &= ~wall_mask_d_3d
861         repeat += 1
862     else:
863         count_last = len(indexes)
864         repeat = 0
865
866     if debug:
867         fig = plt.figure(figsize=(6, 4), dpi=300)
868         plt.axes().axis("off")
869         plt.imshow(zones, cmap="binary")
870         plt.tight_layout()
871
872         fig = plt.figure(figsize=(6, 4), dpi=300)
873         plt.axes().axis("off")
874         plt.imshow(
875             zones_filled[:, :, 0:3] | color_stacked[:, :, 0:3] & 128,
876             cmap="binary",
877         )
878         plt.tight_layout()
879
880         fig = plt.figure(figsize=(6, 4), dpi=300)
881         plt.axes().axis("off")
882         plt.imshow(
883             zones_filled[:, :, 3:6] | color_stacked[:, :, 3:6] & 128,
884             cmap="binary",
885         )
886         plt.tight_layout()
887
888     if repeat == 10:
889         break
890
891     if debug:
892         fig = plt.figure(figsize=(6, 4), dpi=300)
893         plt.axes().axis("off")
894         plt.imshow(zones_filled[:, :, 0:3], cmap="binary")
895         plt.tight_layout()
896
897         fig = plt.figure(figsize=(6, 4), dpi=300)
898         plt.axes().axis("off")
899         plt.imshow(zones_filled[:, :, 3:6], cmap="binary")
900         plt.tight_layout()
901
902     # rollback entrance if it looks too big (sanity check)

```

```

903
904     if debug:
905         fig = plt.figure(figsize=(6, 4), dpi=300)
906         plt.axes().axis("off")
907         plt.imshow(zones_filled[..., 1] & 64 | ent_mask & 128, cmap="binary")
908         plt.tight_layout()
909
910     if cv2.countNonZero(zones_filled[..., 1]) > 1.5 * cv2.countNonZero(ent_mask):
911         zones_filled[..., 1] = ent_mask
912
913     if debug:
914         fig = plt.figure(figsize=(6, 4), dpi=300)
915         plt.axes().axis("off")
916         plt.imshow(zones_filled[:, :, 0:3], cmap="binary")
917         plt.tight_layout()
918
919     ### return wall instead of outdoor
920     unit_comb = np.concatenate(
921         (
922             np.expand_dims(
923                 wall_mask
924                 & cv2.dilate(
925                     np.bitwise_or.reduce(zones_filled[:, :, 1:6], 2), kernel_15c
926                 ),
927                 axis=2,
928             ),
929             zones_filled[:, :, 1:6],
930         ),
931         axis=-1,
932     )
933
934     ### return outdoor/entrance/LDK/bedroom/balcony/bathroom stacked mask
935     return unit_comb
936
937
938 # # test and vis
939
940 # In[6]:
941
942
943 cv2.__version__
944
945
946 # In[7]:

```

```
947
948
949 bgr = read_bgr_from_image_unicode("/fp_img/23776_103B.jpg")
950 # 9765_107A
951 # 1776_105
952 # 102487_266B
953 # 2672_162
954 # 16429_107
955
956 # 현관
957 # 643_105B
958 # 8468_113
959 # 9926_93
960 # 16519_261.jpeg
961 # 20711_137
962 # 14534_178C
963 # 23776_103B
964
965
966 # In[8]:
967
968
969 unit_comb = get_unit_mask(bgr)
970
971
972 # In[9]:
973
974
975 np.amax(unit_comb), np.amin(unit_comb), unit_comb.shape, unit_comb.dtype
976
977
978 # In[10]:
979
980
981 for i in range(6):
982     plt.figure()
983     plt.imshow(unit_comb[:, :, i])
984
```

## 11\_floorplan\_normalization.py

```
1  #!/usr/bin/env python
2  # coding: utf-8
3
4  # In[1]:
5
6
7  get_ipython().run_line_magic("matplotlib", "inline")
8
9  import cv2, matplotlib
10 import numpy as np
11 from skimage.morphology import (
12     skeletonize,
13     skeletonize_3d,
14     medial_axis,
15     thin,
16     local_minima,
17     local_maxima,
18 )
19 from skimage.transform import rescale, resize, downscale_local_mean
20 from scipy.ndimage import distance_transform_edt
21
22 from math import sqrt
23
24 import matplotlib.pyplot as plt
25
26 from os.path import expanduser, splitext
27 from os import scandir, makedirs
28
29 # import random
30
31 import csv
32
33 from tqdm import trange, tqdm_notebook
34
35 from pathlib import Path
36
37 debug = True # plot every steps
38
39
40 # # import floorplan analysis
41
42 # In[2]:
43
```

```

44
45  from floorplan_analysis import read_bgr_from_image_unicode, get_unit_mask
46  from floorplan_analysis import rescale_fp
47  from floorplan_analysis import mono_fp, fp_float_from_mono, fp_uint8_from_mono
48  from floorplan_analysis import pad_fp
49
50
51  # # normalization
52
53  # In[3]:
54
55
56  import pandas as pd
57
58  path_csv = "fp_refined.csv"
59
60  df = pd.read_csv(path_csv)
61  df = df.set_index("id_after")
62  df.Area
63
64
65  # In[85]:
66
67
68  id_ = "27297_131"
69  bgr = read_bgr_from_image_unicode(f"/fp_img/{id_}.jpg")
70
71  # 101160_113E 85
72  # 103915_112C 85
73  # 104127_107B 80
74  # 107903_113G 85
75  # 108838_117B 85
76
77  # 방향 오류 (였던 것)
78  # 1998_53B
79  # 13193_113C
80  # 27874_132A
81  # 100105_144F1
82  # 27297_131
83
84  # 정상
85  # 19778_43
86  # 422_58
87  # 105975_42C12

```

```

88
89  # 밸코니 없음
90  # 5520_150
91  # 101305_96T
92
93  # 응인
94  # 27677_169
95
96
97 plt.imshow(cv2.cvtColor(bgr, cv2.COLOR_BGR2RGB))
98
99
100 # In[86]:
101
102 df.Area[id_]
103
104
105
106 # In[87]:
107
108
109 unit_comb_orig = get_unit_mask(bgr)
110
111
112 # In[88]:
113
114
115 fp = rescale_fp(unit_comb_orig, df.Area[id_])
116 fp = mono_fp(fp)
117 fp = fp_uint8_from_mono(fp)
118
119
120 # In[89]:
121
122
123 fp.shape
124
125
126 # In[90]:
127
128
129 fp.sum(axis=(0, 1))
130
131

```

```

132 # In[91]:
133
134
135 for i in range(0, fp.shape[2], 3):
136     plt.figure()
137     plt.imshow(fp[..., i : i + 3])
138
139
140 # In[92]:
141
142
143 ret, markers = cv2.connectedComponents(fp[:, :, 4])
144 plt.imshow(markers)
145
146
147 # In[93]:
148
149
150 np.unique(markers, return_counts=True)
151
152
153 # In[94]:
154
155
156 d = 11
157 kernel = np.zeros((d, d, 4), np.uint8)
158 kernel[d // 2, : d // 2 + 1, 0] = 1 # right, means facing left
159 kernel[d // 2, d // 2 :, 1] = 1 # left, means facing right
160 kernel[: d // 2 + 1, d // 2, 2] = 1 # bottom, means facing top
161 kernel[d // 2 :, d // 2, 3] = 1 # top, means facing bottom
162
163 kernel[..., 1]
164
165
166 # In[95]:
167
168
169 dilated = np.zeros(markers.shape + (4,), np.uint8)
170 for i in range(4):
171     dilated[..., i] = cv2.dilate(markers.astype(np.uint8), kernel[..., i])
172
173 plt.imshow(dilated[..., 3])
174
175

```

```

176 # 밸코니가 달리는 공간: LDK와 침실 (벽체, 현관, 화장실은 아님)
177
178 # In[96]:
179
180
181 plt.imshow(np.bitwise_or.reduce(fp[..., 2:4], 2))
182
183
184 # In[97]:
185
186
187 adjacent = dilated & np.expand_dims(np.bitwise_or.reduce(fp[..., 2:4], 2), 2)
188 plt.imshow(adjacent[..., 3]) # top, means facing bottom
189
190
191 # In[98]:
192
193
194 plt.imshow(adjacent[..., 2]) # bottom, means facing top
195
196
197 # m00 면적, nu20 가로 길이, nu02 세로 길이
198
199 # In[99]:
200
201
202 cv2.moments((markers == 1).astype(int), True)
203
204
205 # 크기와 무관한 nu20 vs nu02
206 #
207 # 대각선은 거의 1:1 (0.133 vs 0.124)
208 #
209 # 짧아도 4배 이상은 차이 나고 거의 정사각형 같아야 1.5배 차이
210 # 2배로 잡으면 적당할 듯
211
212 # In[100]:
213
214
215 list(range(1, ret))
216
217
218 # In[101]:
219

```

```

220
221 moments = cv2.moments((markers == 1).astype(int), True)
222 moments["nu20"], moments["nu02"]
223
224
225 # In[102]:
226
227
228 (adjacent[..., 3] == 5).sum()
229
230
231 # In[103]:
232
233
234 cv2.countNonZero((adjacent[..., 3] == 1).astype(np.uint8))
235
236
237 # In[104]:
238
239
240 if moments["nu20"] / moments["nu02"] > 2:
241     # horizontal = top or bottom
242     if (adjacent[..., 2] == 1).sum() > (adjacent[..., 3] == 1).sum():
243         # facing top
244         print("top")
245     else:
246         # facing bottom
247         print("bottom")
248
249
250 # In[105]:
251
252
253 (markers == 1).shape
254
255
256 # In[106]:
257
258
259 np.bitwise_or.reduce((markers == 1), 0) # reduce row = y
260
261
262 # In[107]:
263
```

```

264
265     np.bitwise_or.reduce((markers == 1), 0).sum(), np.bitwise_or.reduce(
266         (markers == 1), 1
267     ).sum()
268
269
270 # In[108]:
271
272
273 sum_by_facing = [0, 0, 0, 0]
274 for i in range(1, ret):
275     moments = cv2.moments((markers == i).astype(int), True)
276     aspect = moments["mu20"] / moments["mu02"]
277     if aspect > 2:
278         # horizontal = top or bottom
279         width = np.bitwise_or.reduce((markers == i), 0).sum()
280
281         if (adjacent[..., 2] == i).sum() > (adjacent[..., 3] == i).sum():
282             # facing top
283             sum_by_facing[2] += width
284         else:
285             # facing bottom
286             sum_by_facing[3] += width
287     elif aspect < 0.5:
288         # vertical = left or right
289         width = np.bitwise_or.reduce((markers == i), 1).sum()
290
291         if (adjacent[..., 0] == i).sum() > (adjacent[..., 1] == i).sum():
292             # facing left
293             sum_by_facing[0] += width
294         else:
295             # facing right
296             sum_by_facing[1] += width
297     else:
298         # ignore
299         pass
300 print(sum_by_facing) # lrtb order
301
302
303 # In[109]:
304
305
306 # remove minor facing
307

```

```

308 sum_by_facing = [
309     0 if width < max(sum_by_facing) / 2 else width for width in sum_by_facing
310 ]
311 sum_by_facing
312
313
314 # # position of main bedroom
315
316 # In[110]:
317
318
319 dist_transform = cv2.distanceTransform(fp[:, :, 3], cv2.DIST_L2, 5)
320 plt.imshow(dist_transform)
321
322
323 # In[111]:
324
325
326 dist_transform.max()
327
328
329 # In[112]:
330
331
332 mbr_core = (dist_transform == dist_transform.max()).astype(np.uint8)
333 plt.imshow(mbr_core)
334
335
336 # In[113]:
337
338
339 plt.imshow(np.bitwise_or.reduce(fp[..., 1:], 2))
340
341
342 # In[114]:
343
344
345 moments = cv2.moments(np.bitwise_or.reduce(fp[..., 1:], 2), True)
346 center_h = round(moments["m10"] / moments["m00"]) # center of floorplan
347 center_h
348
349
350 # In[115]:
351

```

```

352
353 moments = cv2.moments(np.bitwise_or.reduce(fp[...], 1:], 2), True)
354 center_v = round(moments["m01"] / moments["m00"]) # center of floorplan
355 center_v
356
357
358 # In[116]:
359
360
361 mbr_core.shape, mbr_core[:, :center_h].shape, mbr_core[:, center_h:].shape
362
363
364 # In[117]:
365
366
367 mbr_core[:, :center_h].sum(), mbr_core[:, center_h:].sum()
368
369
370 # In[118]:
371
372
373 mbr_core[:center_v, :].sum(), mbr_core[center_v:, :].sum() # top, bottom
374
375
376 # In[119]:
377
378
379 mbr_left = mbr_core[:, :center_h].sum()
380 mbr_right = mbr_core[:, center_h:].sum()
381
382 if mbr_left > mbr_right:
383     print("left")
384 elif mbr_left < mbr_right:
385     print("right")
386 else:
387     print("error")
388
389
390 # In[120]:
391
392
393 def mbr_position(fp):
394     # center of floorplan
395     moments = cv2.moments(np.bitwise_or.reduce(fp[...], 1:], 2), True)

```

```

396     center = (
397         round(moments["m10"] / moments["m00"]), # x
398         round(moments["m01"] / moments["m00"]), # y
399     )
400
401     # core of main bedroom
402     dist_transform = cv2.distanceTransform(fp[:, :, 3], cv2.DIST_L2, 5)
403     mbr_core = (dist_transform == dist_transform.max()).astype(np.uint8)
404
405     # horizontal
406     mbr_left = mbr_core[:, : center[0]].sum()
407     mbr_right = mbr_core[:, center[0] :].sum()
408
409     if mbr_left > mbr_right:
410         horizontal = "left"
411     elif mbr_left < mbr_right:
412         horizontal = "right"
413     else:
414         horizontal = "not sure"
415
416     # vertical
417     mbr_top = mbr_core[: center[1], :].sum()
418     mbr_bottom = mbr_core[center[1] :, :].sum()
419
420     if mbr_top > mbr_bottom:
421         vertical = "top"
422     elif mbr_top < mbr_bottom:
423         vertical = "bottom"
424     else:
425         vertical = "not sure"
426
427     return horizontal, vertical
428
429
430 # In[121]:
431
432
433 mbr_position(fp)
434
435
436 # # pick one from opposite facings
437
438 # In[122]:
439
```

```

440
441     horizontal, vertical = mbr_position(fp)
442
443     if sum_by_facing[0] and sum_by_facing[1]: # if left and right are both valid
444         if horizontal == "left":
445             sum_by_facing[1] = 0
446         elif horizontal == "right":
447             sum_by_facing[0] = 0
448         else:
449             if sum_by_facing[0] > sum_by_facing[1]:
450                 sum_by_facing[1] = 0
451             else:
452                 sum_by_facing[0] = 0
453
454     if sum_by_facing[2] and sum_by_facing[3]: # if top and bottom are both valid
455         if vertical == "top":
456             sum_by_facing[3] = 0
457         elif vertical == "bottom":
458             sum_by_facing[2] = 0
459         else:
460             if sum_by_facing[2] > sum_by_facing[3]:
461                 sum_by_facing[3] = 0
462             else:
463                 sum_by_facing[2] = 0
464
465
466     # In[123]:
467
468
469     sum_by_facing
470
471
472     # In[124]:
473
474
475     ["left", "right", "top", "bottom"][sum_by_facing.index(max(sum_by_facing))]
476
477
478     # # facing detection by balcony, final
479
480     # In[125]:
481
482
483     def balcony_per_facing(fp):

```

```

484     """balcony width by facing
485     returns list of total widths of balconies
486     facing left, right, top, bottom"""
487
488     # get connected balcony zones
489
490     ret, markers = cv2.connectedComponents(fp[:, :, 4])
491
492     # get adjacent areas per balcony
493
494     d = 11
495     kernel = np.zeros((d, d, 4), np.uint8)
496     kernel[d // 2, : d // 2 + 1, 0] = 1 # right, means facing left
497     kernel[d // 2, d // 2 :, 1] = 1 # left, means facing right
498     kernel[: d // 2 + 1, d // 2, 2] = 1 # bottom, means facing top
499     kernel[d // 2 :, d // 2, 3] = 1 # top, means facing bottom
500
501     dilated = np.zeros(markers.shape + (4,), np.uint8)
502     for i in range(4):
503         dilated[..., i] = cv2.dilate(markers.astype(np.uint8), kernel[..., i])
504     adjacent = dilated & np.expand_dims(np.bitwise_or.reduce(fp[..., 2:4], 2), 2)
505
506     # get sum by facing
507
508     sum_by_facing = [0, 0, 0, 0]
509     for i in range(1, ret):
510         moments = cv2.moments((markers == i).astype(int), True)
511         aspect = moments["nu20"] / moments["nu02"]
512         if aspect > 2:
513             # horizontal = top or bottom
514             width = np.bitwise_or.reduce((markers == i), 0).sum()
515
516             if (adjacent[..., 2] == i).sum() > (adjacent[..., 3] == i).sum():
517                 # facing top
518                 sum_by_facing[2] += width
519             else:
520                 # facing bottom
521                 sum_by_facing[3] += width
522         elif aspect < 0.5:
523             # vertical = left or right
524             width = np.bitwise_or.reduce((markers == i), 1).sum()
525
526             if (adjacent[..., 0] == i).sum() > (adjacent[..., 1] == i).sum():
527                 # facing left

```

```

528             sum_by_facing[0] += width
529         else:
530             # facing right
531             sum_by_facing[1] += width
532     else:
533         # ignore
534         pass
535
536     return sum_by_facing # lrtb
537
538
539 def facing_by_balcony(sum_by_facing, mbr_position):
540
541     assert any(sum_by_facing), "no balcony"
542
543     # remove minor facing
544
545     sum_by_facing = [
546         0 if width < max(sum_by_facing) / 2 else width for width in sum_by_facing
547     ]
548
549     # remove opposite
550
551     horizontal, vertical = mbr_position
552
553     if sum_by_facing[0] and sum_by_facing[1]: # if left and right are both valid
554         if horizontal == "left":
555             sum_by_facing[1] = 0
556         elif horizontal == "right":
557             sum_by_facing[0] = 0
558         else:
559             if sum_by_facing[0] > sum_by_facing[1]:
560                 sum_by_facing[1] = 0
561             else:
562                 sum_by_facing[0] = 0
563
564     if sum_by_facing[2] and sum_by_facing[3]: # if top and bottom are both valid
565         if vertical == "top":
566             sum_by_facing[3] = 0
567         elif vertical == "bottom":
568             sum_by_facing[2] = 0
569         else:
570             if sum_by_facing[2] > sum_by_facing[3]:
571                 sum_by_facing[3] = 0

```

```

572         else:
573             sum_by_facing[2] = 0
574
575     # return facing with most balcony
576     return sum_by_facing.index(max(sum_by_facing)) # lrtb
577
578
579 # In[126]:
580
581
582 # 방향 오류 (였던 것)
583 # 1998_53B
584 # 13193_113C
585 # 27874_132A
586 # 100105_144F1
587 # 27297_131
588
589 # 정상
590 # 19778_43
591 # 422_58
592 # 105975_42C12
593
594
595 # In[127]:
596
597
598 id_ = "13193_113C"
599 bgr = read_bgr_from_image_unicode(f"/fp_img/{id_}.jpg")
600 unit_comb_orig = get_unit_mask(bgr)
601 fp = rescale_fp(unit_comb_orig, df.Area[id_])
602 fp = mono_fp(fp)
603 fp = fp_uint8_from_mono(fp)
604
605 plt.imshow(cv2.cvtColor(bgr, cv2.COLOR_BGR2RGB))
606
607 bal_per_facing = balcony_per_facing(fp)
608 if any(bal_per_facing):
609     facing = facing_by_balcony(bal_per_facing, mbr_position(fp))
610     print(["left", "right", "top", "bottom"][facing])
611 else:
612     print("no balcony")
613
614
615 # # no balcony

```

```

616
617 # In[47]:
618
619
620 # 밸코니 없음
621 # 5520_150
622 # 101305_96T
623 # 104168_116B
624 # 106068_31H1
625 # 106449_41E
626 # 109582_52C
627
628 # 소형
629 # 422_58
630 # 8844_57
631
632
633 # In[48]:
634
635
636 id_ = "104168_116B"
637 bgr = read_bgr_from_image_unicode(f"/fp_img/{id_}.jpg")
638 unit_comb_orig = get_unit_mask(bgr)
639 fp = rescale_fp(unit_comb_orig, df.Area[id_])
640 fp = mono_fp(fp)
641 fp = fp_uint8_from_mono(fp)
642
643 plt.imshow(cv2.cvtColor(bgr, cv2.COLOR_BGR2RGB))
644
645
646 # In[49]:
647
648
649 d = 5
650 kernel = np.zeros((d, d, 2), np.uint8)
651 kernel[d // 2, :, 0] = 1 # horizontal, for horizontal walls, means facing left or right
652 kernel[:, d // 2, 1] = 1 # vertical, for vertical walls, means facing top or bottom
653
654
655 print(kernel[..., 0])
656 print(kernel[..., 1])
657
658
659 # In[50]:

```

```
660
661
662 erosion = np.zeros(fp.shape[:2] + (2,), np.uint8)
663 for i in range(2):
664     erosion[..., i] = cv2.erode(fp[..., 0], kernel[..., i])
665
666
667 # In[51]:
668
669
670 plt.imshow(fp[... , 0])
671
672
673 # In[52]:
674
675
676 plt.imshow(erosion[... , 0])
677
678
679 # In[53]:
680
681
682 plt.imshow(erosion[... , 1])
683
684
685 # In[54]:
686
687
688 plt.imshow(erosion[... , 0] & 128 | erosion[... , 1])
689
690
691 # In[55]:
692
693
694 cv2.countNonZero(erosion[... , 0]), cv2.countNonZero(erosion[... , 1])
695
696
697 # In[56]:
698
699
700 plt.imshow(skeletonize(erosion[... , 0] > 0))
701
702
703 # In[57]:
```

```

704
705
706 skeletonize(erosion[..., 0] > 0).sum()
707
708
709 # In[58]:
710
711
712 plt.imshow(skeletonize(erosion[..., 1] > 0))
713
714
715 # In[59]:
716
717
718 skeletonize(erosion[..., 1] > 0).sum()
719
720
721 # In[60]:
722
723
724 ["left/right", "top/bottom"][
725     0
726     if skeletonize(erosion[..., 0] > 0).sum() > skeletonize(erosion[..., 1] > 0).sum()
727     else 1
728 ]
729
730
731 # In[61]:
732
733
734 def ent_position(fp):
735     # center of floorplan
736     moments = cv2.moments(np.bitwise_or.reduce(fp[..., 1:], 2), True)
737     center = (
738         round(moments["m10"] / moments["m00"]), # x
739         round(moments["m01"] / moments["m00"]), # y
740     )
741
742     # core of entrance
743     dist_transform = cv2.distanceTransform(fp[:, :, 1], cv2.DIST_L2, 5)
744     core = (dist_transform == dist_transform.max()).astype(np.uint8)
745
746     # horizontal
747     left = core[:, : center[0]].sum()

```

```

748     right = core[:, center[0] :].sum()
749
750     if left > right:
751         horizontal = "left"
752     elif left < right:
753         horizontal = "right"
754     else:
755         horizontal = "not sure"
756
757     # vertical
758     top = core[: center[1], :].sum()
759     bottom = core[center[1] :, :].sum()
760
761     if top > bottom:
762         vertical = "top"
763     elif top < bottom:
764         vertical = "bottom"
765     else:
766         vertical = "not sure"
767
768     return horizontal, vertical
769
770
771 # In[62]:
772
773
774 ent_position(fp)
775
776
777 # In[63]:
778
779
780 def facing_by_wall(fp):
781     d = 5
782     kernel = np.zeros((d, d, 2), np.uint8)
783     kernel[
784         d // 2, :, 0
785     ] = 1 # horizontal, for horizontal walls, means facing left or right
786     kernel[:, d // 2, 1] = 1 # vertical, for vertical walls, means facing top or bottom
787
788     erosion = np.zeros(fp.shape[:2] + (2,), np.uint8)
789     for i in range(2):
790         erosion[..., i] = cv2.erode(fp[..., 0], kernel[..., i])
791

```

```

792     wall_h = skeletonize(erosion[..., 0] > 0).sum()
793     wall_v = skeletonize(erosion[..., 1] > 0).sum()
794
795     horizontal, vertical = ent_position(fp)
796     if wall_h > wall_v:
797         # left or right
798         if horizontal == "left":
799             facing = 1 # right
800         elif horizontal == "right":
801             facing = 0 # left
802         else:
803             facing = -1 # not sure
804     else:
805         # top or bottom
806         if vertical == "top":
807             facing = 3 # top
808         elif vertical == "bottom":
809             facing = 2 # bottom
810         else:
811             facing = -1 # not sure
812
813     return facing # lrtb
814
815
816 # In[64]:
817
818
819 ["left", "right", "top", "bottom"][facing_by_wall(fp)]
820
821
822 # In[65]:
823
824
825 def align_fp(fp): # rescale first
826     """put the main side to down and entrance to left"""
827
828     def mbr_position(fp):
829         # center of floorplan
830         moments = cv2.moments(np.bitwise_or.reduce(fp[...,:], 2), True)
831         center = (
832             round(moments["m10"] / moments["m00"]), # x
833             round(moments["m01"] / moments["m00"])), # y
834         )
835

```

```

836     # core of main bedroom
837     dist_transform = cv2.distanceTransform(fp[:, :, 3], cv2.DIST_L2, 5)
838     mbr_core = (dist_transform == dist_transform.max()).astype(np.uint8)
839
840     # horizontal
841     mbr_left = mbr_core[:, : center[0]].sum()
842     mbr_right = mbr_core[:, center[0] :].sum()
843
844     if mbr_left > mbr_right:
845         horizontal = "left"
846     elif mbr_left < mbr_right:
847         horizontal = "right"
848     else:
849         horizontal = "not sure"
850
851     # vertical
852     mbr_top = mbr_core[: center[1], :].sum()
853     mbr_bottom = mbr_core[center[1] :, :].sum()
854
855     if mbr_top > mbr_bottom:
856         vertical = "top"
857     elif mbr_top < mbr_bottom:
858         vertical = "bottom"
859     else:
860         vertical = "not sure"
861
862     return horizontal, vertical
863
864 def ent_position(fp):
865     # center of floorplan
866     moments = cv2.moments(np.bitwise_or.reduce(fp[...], 1), True)
867     center = (
868         round(moments["m10"] / moments["m00"]), # x
869         round(moments["m01"] / moments["m00"]), # y
870     )
871
872     # core of entrance
873     dist_transform = cv2.distanceTransform(fp[:, :, 1], cv2.DIST_L2, 5)
874     core = (dist_transform == dist_transform.max()).astype(np.uint8)
875
876     # horizontal
877     left = core[:, : center[0]].sum()
878     right = core[:, center[0] :].sum()
879

```

```

880         if left > right:
881             horizontal = "left"
882         elif left < right:
883             horizontal = "right"
884         else:
885             horizontal = "not sure"
886
887         # vertical
888         top = core[: center[1], :].sum()
889         bottom = core[center[1] :, :].sum()
890
891         if top > bottom:
892             vertical = "top"
893         elif top < bottom:
894             vertical = "bottom"
895         else:
896             vertical = "not sure"
897
898         return horizontal, vertical
899
900     def balcony_per_facing(fp):
901         """balcony width by facing
902         returns list of total widths of balconies
903         facing left, right, top, bottom"""
904
905         # get connected balcony zones
906
907         ret, markers = cv2.connectedComponents(fp[:, :, 4])
908
909         # get adjacent areas per balcony
910
911         d = 11
912         kernel = np.zeros((d, d, 4), np.uint8)
913         kernel[d // 2, : d // 2 + 1, 0] = 1 # right, means facing left
914         kernel[d // 2, d // 2 :, 1] = 1 # left, means facing right
915         kernel[:, d // 2 + 1, d // 2, 2] = 1 # bottom, means facing top
916         kernel[d // 2 :, d // 2, 3] = 1 # top, means facing bottom
917
918         dilated = np.zeros(markers.shape + (4,), np.uint8)
919         for i in range(4):
920             dilated[..., i] = cv2.dilate(markers.astype(np.uint8), kernel[..., i])
921         adjacent = dilated & np.expand_dims(np.bitwise_or.reduce(fp[..., 2:4], 2), 2)
922
923         # get sum by facing

```

```

924
925     sum_by_facing = [0, 0, 0, 0]
926     for i in range(1, ret):
927         moments = cv2.moments((markers == i).astype(int), True)
928         aspect = moments["nu20"] / moments["nu02"]
929         if aspect > 2:
930             # horizontal = top or bottom
931             width = np.bitwise_or.reduce((markers == i), 0).sum()
932
933             if (adjacent[..., 2] == i).sum() > (adjacent[..., 3] == i).sum():
934                 # facing top
935                 sum_by_facing[2] += width
936             else:
937                 # facing bottom
938                 sum_by_facing[3] += width
939         elif aspect < 0.5:
940             # vertical = left or right
941             width = np.bitwise_or.reduce((markers == i), 1).sum()
942
943             if (adjacent[..., 0] == i).sum() > (adjacent[..., 1] == i).sum():
944                 # facing left
945                 sum_by_facing[0] += width
946             else:
947                 # facing right
948                 sum_by_facing[1] += width
949         else:
950             # ignore
951             pass
952
953     return sum_by_facing # lrtb
954
955 def facing_by_balcony(sum_by_facing, mbr_position):
956     assert any(sum_by_facing), "no balcony"
957
958     # remove minor facing
959
960     sum_by_facing = [
961         0 if width < max(sum_by_facing) / 2 else width for width in sum_by_facing
962     ]
963
964     # remove opposite
965
966     horizontal, vertical = mbr_position
967

```

```

968     if sum_by_facing[0] and sum_by_facing[1]: # if left and right are both valid
969         if horizontal == "left":
970             sum_by_facing[1] = 0
971         elif horizontal == "right":
972             sum_by_facing[0] = 0
973         else:
974             if sum_by_facing[0] > sum_by_facing[1]:
975                 sum_by_facing[1] = 0
976             else:
977                 sum_by_facing[0] = 0
978
979     if sum_by_facing[2] and sum_by_facing[3]: # if top and bottom are both valid
980         if vertical == "top":
981             sum_by_facing[3] = 0
982         elif vertical == "bottom":
983             sum_by_facing[2] = 0
984         else:
985             if sum_by_facing[2] > sum_by_facing[3]:
986                 sum_by_facing[3] = 0
987             else:
988                 sum_by_facing[2] = 0
989
990     # return facing with most balcony
991     return sum_by_facing.index(max(sum_by_facing)) # lrtb
992
993 def facing_by_wall(fp):
994     d = 5
995     kernel = np.zeros((d, d, 2), np.uint8)
996     kernel[
997         d // 2, :, 0
998     ] = 1 # horizontal, for horizontal walls, means facing left or right
999     kernel[
1000         :, d // 2, 1
1001     ] = 1 # vertical, for vertical walls, means facing top or bottom
1002
1003     erosion = np.zeros(fp.shape[:2] + (2,), np.uint8)
1004     for i in range(2):
1005         erosion[..., i] = cv2.erode(fp[..., 0], kernel[..., i])
1006
1007     wall_h = skeletonize(erosion[..., 0] > 0).sum()
1008     wall_v = skeletonize(erosion[..., 1] > 0).sum()
1009
1010     horizontal, vertical = ent_position(fp)
1011     if wall_h > wall_v:

```

```

1012         # left or right
1013         if horizontal == "left":
1014             # facing right
1015             facing = 1
1016         elif horizontal == "right":
1017             # facing left
1018             facing = 0
1019         else:
1020             facing = -1 # not sure
1021     else:
1022         # top or bottom
1023         if vertical == "top":
1024             # facing bottom
1025             facing = 3
1026         elif vertical == "bottom":
1027             # facing top
1028             facing = 2
1029         else:
1030             facing = -1 # not sure
1031
1032     return facing # lrtn
1033
1034     ### process
1035
1036     bal_per_facing = balcony_per_facing(fp)
1037     if any(bal_per_facing):
1038         facing = facing_by_balcony(bal_per_facing, mbr_position(fp))
1039     else:
1040         facing = facing_by_wall(fp)
1041
1042     if facing > 0:
1043         print(["left", "right", "top", "bottom"][facing])
1044     else:
1045         # not sure, don't do anything
1046         pass
1047
1048     if facing == 0:
1049         # facing left
1050         fp = np.rot90(fp, 1)
1051     elif facing == 1:
1052         # facing right
1053         fp = np.rot90(fp, -1)
1054     elif facing == 2:
1055         # facing top

```

```

1056         fp = np.rot90(fp, 2)
1057     elif facing == 3:
1058         # facing bottom, don't do anything
1059         pass
1060     else:
1061         # not sure, don't do anything
1062         pass
1063
1064     # put entrance to left
1065     horizontal, _ = ent_position(fp)
1066     if horizontal == "right":
1067         fp = np.flip(fp, axis=1)
1068
1069     return fp
1070
1071
1072 # In[66]:
1073
1074
1075 plt.imshow(align_fp(fp)[..., 0:3])
1076
1077
1078 # # rescale
1079
1080 # In[67]:
1081
1082
1083 area = 85.0
1084 target_ppm = 5 # pixels per meter
1085
1086 # indoor pixels excluding balcony
1087 pixels = cv2.countNonZero(np.bitwise_or.reduce(fp, 2) & ~fp[:, :, 4])
1088
1089 print(area, pixels)
1090
1091 scale = sqrt(area * target_ppm ** 2 / pixels)
1092 print(scale)
1093
1094 unit_scale = rescale(fp, scale, mode="edge", multichannel=True)
1095 plt.imshow(unit_scale[:, :, 0])
1096
1097
1098 # In[68]:
1099
```

```

1100
1101     indexes = np.where(unit_scale != 0)
1102
1103
1104     # In[69]:
1105
1106
1107     unit_clipped = unit_scale[
1108         min(indexes[0]) : max(indexes[0]) + 1, min(indexes[1]) : max(indexes[1]) + 1
1109     ]
1110     plt.imshow(unit_clipped[:, :, 4])
1111
1112
1113     # In[70]:
1114
1115
1116     from skimage.transform import rescale
1117
1118
1119     def rescale_fp(unit_comb, area, target_ppm=5, trim=True):
1120         # indoor pixels excluding balcony
1121         pixels = cv2.countNonZero(np.bitwise_or.reduce(unit_comb, 2) & ~unit_comb[:, :, 4])
1122         assert pixels > 0
1123
1124         scale = sqrt(area * target_ppm ** 2 / pixels)
1125
1126         unit_scale = rescale(unit_comb, scale, mode="edge", multichannel=True)
1127
1128         if trim:
1129             indexes = np.where(unit_scale != 0)
1130             unit_scale = unit_scale[
1131                 min(indexes[0]) : max(indexes[0]) + 1, min(indexes[1]) : max(indexes[1]) + 1
1132             ]
1133
1134         return (unit_scale * 255).astype(np.uint8)
1135
1136
1137     # rescale first

```

## 12\_floorplan\_image.py

```
1  #!/usr/bin/env python
2  # coding: utf-8
3
4  # In[1]:
5
6
7  get_ipython().run_line_magic("matplotlib", "inline")
8
9  import cv2, matplotlib
10 import numpy as np
11 from skimage.morphology import (
12     skeletonize,
13     skeletonize_3d,
14     medial_axis,
15     thin,
16     local_minima,
17     local_maxima,
18 )
19 from skimage.transform import rescale, resize, downscale_local_mean
20 from scipy.ndimage import distance_transform_edt
21
22 from math import sqrt, log10
23
24 import matplotlib.pyplot as plt
25
26 from os.path import expanduser, splitext
27 from os import scandir, makedirs
28
29 # import random
30
31 import csv
32
33 from tqdm import trange, tqdm_notebook
34
35 from pathlib import Path
36
37 debug = False # plot every steps
38
39
40 # In[2]:
41
42
43 from floorplan_analysis import read_bgr_from_image_unicode, get_unit_mask
```

```
44 from floorplan_analysis import align_fp, rescale_fp
45
46
47 # In[3]:
48
49
50 cv2.__version__
51
52
53 # In[4]:
54
55
56 bgr = read_bgr_from_image_unicode("/fp_img/10001_57B.jpg")
57 # 9765_107A 누워있는
58 # 1776_105 코어
59 # 102487_266B 비사각
60
61 # 199_86
62 # 6_87
63 # 2_63
64 # 8_99
65
66 # 107323_110B 흰 영역 날아감
67
68 # 10001_57B 밤크니 없음
69
70
71 # In[5]:
72
73
74 plt.imshow(bgr)
75
76
77 # In[6]:
78
79
80 unit_comb_orig = get_unit_mask(bgr)
81
82
83 # In[7]:
84
85
86 plt.imshow(np.bitwise_or.reduce(unit_comb_orig, 2))
87
```

```

88
89 # In[8]:
90
91
92 cv2.countNonZero(unit_comb_orig[:, :, 4])
93
94
95 # In[9]:
96
97
98 unit_comb = rescale_fp(unit_comb_orig.copy(), 85)
99 unit_comb = align_fp(unit_comb)
100
101 plt.imshow(unit_comb[:, :, [0, 1, 4]])
102
103
104 # In[ ]:
105
106
107 # In[ ]:
108
109
110 # In[10]:
111
112
113 # AREA_WALL = 64
114 # AREA_ENTRANCE = 32
115 # AREA_LDK = 16
116 # AREA_BEDROOM = 8
117 # AREA_BALCONY = 4
118 # AREA_BATHROOM = 2
119
120 mask_bits = np.array([64, 32, 16, 8, 4, 2], dtype=np.uint8)
121
122
123 # In[11]:
124
125
126 # binary cut value
127 cut = 0.01 * np.ones(6)
128 for i in range(6):
129     if cv2.countNonZero(unit_comb[:, :, i]):
130         cut[i] = (
131             np.average(unit_comb[:, :, i][np.nonzero(unit_comb[:, :, i] > 0)]) - 0.01

```

```

132         )
133
134
135 # In[12]:
136
137
138 plt.imshow((unit_comb > cut)[:, :, 4])
139
140
141 # In[13]:
142
143
144 mono = ((unit_comb > cut) * 255) & mask_bits
145
146
147 # In[14]:
148
149
150 mono.shape, mono.dtype
151
152
153 # In[15]:
154
155
156 npamax(mono[:, :, 2])
157
158
159 # In[16]:
160
161
162 mono = np.bitwise_or.reduce(mono, 2)
163 plt.imshow(mono)
164
165
166 # In[17]:
167
168
169 def mono_fp(unit_comb):
170     """create bit mask image from
171     wall/entrance/LDK/bedroom/balcony/bathroom stacked array"""
172
173     # AREA_WALL = 64
174     # AREA_ENTRANCE = 32
175     # AREA_LDK = 16

```

```

176     # AREA_BEDROOM = 8
177     # AREA_BALCONY = 4
178     # AREA_BATHROOM = 2
179
180     mask_bits = np.array([64, 32, 16, 8, 4, 2], dtype=np.uint8)
181
182     # binary cut value
183     cut = 0.01 * np.ones(6)
184     for i in range(6):
185         if cv2.countNonZero(unit_comb[:, :, i]):
186             cut[i] = (
187                 np.average(unit_comb[:, :, i][np.nonzero(unit_comb[:, :, i] > 0)])
188                 - 0.01
189             )
190
191     mono = ((unit_comb > cut) * 255).astype(np.uint8) & mask_bits
192     mono = np.bitwise_or.reduce(mono, 2)
193     return mono

```

## 15\_process\_floorplan\_images.py

```
1  #!/usr/bin/env python
2  # coding: utf-8
3
4  # In[1]:
5
6
7  get_ipython().run_line_magic("matplotlib", "inline")
8
9  import cv2, matplotlib
10 import numpy as np
11 from skimage.morphology import (
12     skeletonize,
13     skeletonize_3d,
14     medial_axis,
15     thin,
16     local_minima,
17     local_maxima,
18 )
19 from skimage.transform import rescale, resize, downscale_local_mean
20 from scipy.ndimage import distance_transform_edt
21
22 from math import sqrt
23
24 import matplotlib.pyplot as plt
25
26 from os.path import expanduser, splitext
27 from os import scandir, makedirs
28
29 # import random
30
31 import csv
32
33 from tqdm import trange, tqdm_notebook
34
35 from pathlib import Path
36
37 debug = False # plot every steps
38
39
40 # In[2]:
41
42
43 from floorplan_analysis import read_bgr_from_image_unicode
```

```

44     from floorplan_analysis import read_from_csv
45
46     from floorplan_analysis import get_unit_mask
47
48     from floorplan_analysis import align_fp, rescale_fp
49     from floorplan_analysis import mono_fp
50     from floorplan_analysis import read_mono_from_image_unicode, save_mono_to_image_unicode
51
52
53     # # process files
54
55     # In[3]:
56
57
58     def process_floorplan_mono(
59         path_from, area, filename_to, dir_to="/data/fp_img_processed/", ext_to=".png"
60     ):
61         try:
62             bgr = read_bgr_from_image_unicode(path_from)
63             unit_comb = get_unit_mask(bgr)
64             unit_comb = rescale_fp(unit_comb, area)
65             unit_comb = align_fp(unit_comb)
66
67             mono = mono_fp(unit_comb)
68             save_mono_to_image_unicode(mono, dir_to + filename_to + ext_to, ext_to)
69         except:
70             print(filename_to)
71
72
73     # In[4]:
74
75
76     process_floorplan_mono("/fp_img/10001_57B.jpg", 85, "mono", dir_to="")
77
78
79     # In[5]:
80
81
82     plt.imshow(read_mono_from_image_unicode("mono.png"))
83
84
85     # In[6]:
86
87

```

```

88 bgr = read_bgr_from_image_unicode("/fp_img/10001_57B.jpg")
89 plt.imshow(bgr)
90
91
92 # In[7]:
93
94
95 unit_comb = get_unit_mask(bgr)
96
97
98 # In[8]:
99
100
101 unit_comb = rescale_fp(unit_comb, 85)
102
103
104 # In[9]:
105
106
107 unit_comb = align_fp(unit_comb)
108 plt.imshow(unit_comb[:, :, 2])
109
110
111 # In[10]:
112
113
114 mono = mono_fp(unit_comb)
115
116
117 # # multiprocessing
118
119 # In[11]:
120
121
122 from multiprocessing import Pool
123
124
125 def worker(x, y):
126     return x * y
127
128
129 with Pool(7) as p:
130     output = p.starmap(worker, [(i, 2 * i) for i in range(101)], chunksize=100)
131

```

```

132     print(output)
133
134
135     # # main
136
137     # In[12]:
138
139
140     dir_ID_from = "/fp_img/"
141     dir_IDS_exclude = "/data/exclude/"
142
143     dir_from = "/fp_img/"
144
145     dir_to = "/data/fp_img_processed/"
146     makedirs(dir_to, exist_ok=True)
147
148     ext_to = ".png"
149
150     ### all of the plans
151     ID_ext_dict = {
152         splitext(f.name)[0]: splitext(f.name)[1]
153         for f in scandir(dir_ID_from)
154         if f.is_file()
155     }
156     print(len(ID_ext_dict.keys()), "floorplans")
157
158
159     # In[13]:
160
161
162     list(ID_ext_dict.items())[:10]
163
164
165     # In[14]:
166
167
168     files_IDS_exclude = list(Path(expanduser(dir_IDS_exclude)).glob("*.csv"))
169
170     # don't repeat the process
171     files_IDS_exclude.append(Path("processed.csv"))
172
173     print(files_IDS_exclude)
174
175

```

```

176 # In[15]:
177
178
179 IDs_excl = set()
180 for file_excl in files_IDs_exclude:
181     _, file_excl_list = read_from_csv(str(file_excl))
182     if file_excl_list:
183         list_excl = [row[0] for row in file_excl_list]
184         IDs_excl |= set(list_excl)
185     print(file_excl, "processed:", len(list_excl), "floorplans to exclude")
186
187 # _, fp_img_processed_list = read_from_csv(exp_path_fp_img)
188 # if fp_img_processed_list:
189 #     list_excl = [row[0] for row in fp_img_processed_list]
190 #     IDs_excl |= set(list_excl)
191 #     print(len(list_excl), "floorplans already processed")
192
193
194 # In[16]:
195
196
197 import pandas as pd
198
199 path_csv = "fp_refined.csv"
200
201 df = pd.read_csv(path_csv)
202 df = df.set_index("id_after")
203 df
204
205
206 # In[17]:
207
208
209 ID_set = set(ID_ext_dict.keys()).difference(IDs_excl)
210 ID_set = ID_set.intersection(df.index)
211 IDs = list(ID_set)
212 print(len(IDs), "floorplans to go")
213
214
215 # In[18]:
216
217
218 paths_from = [dir_from + ID + ID_ext_dict[ID] for ID in IDs]
219 print(paths_from[:10])

```

```
220
221
222 # In[19]:
223
224 df.Area
225
226
227
228 # In[20]:
229
230
231 area_list = [df.Area[ID] for ID in IDs]
232 area_list[:10]
233
234
235 # In[21]:
236
237
238 from multiprocessing import Pool
239
240 makedirs(dir_to, exist_ok=True)
241
242 with Pool(7) as p:
243     p.starmap(process_floorplan_mono, zip(paths_from, area_list, IDs))
244
245
246 #     ls /data/floorplan_mono/ | wc -l
```

## 16\_processed\_list.py

```
1  #!/usr/bin/env python
2  # coding: utf-8
3
4  # In[1]:
5
6
7  get_ipython().run_line_magic("matplotlib", "inline")
8
9  from PIL import Image
10
11 import cv2, matplotlib
12 import numpy as np
13
14 from math import sqrt
15
16 import matplotlib.pyplot as plt
17
18 from os.path import expanduser, splitext
19 from os import scandir, makedirs
20
21 import csv
22
23 from tqdm.notebook import trange, tqdm
24
25 from pathlib import Path
26
27 debug = False # plot every steps
28
29
30 # In[2]:
31
32
33 from floorplan_analysis import read_from_csv
34
35
36 # # CSV
37
38 # In[3]:
39
40
41 dir_from = "/data/fp_img_processed/"
42
43 csv_to = "processed.csv"
```

```

44
45  ### all of the plans
46  ID_path_dict = {splitext(f.name)[0]: f.path for f in scandir(dir_from) if f.is_file()}
47  print(len(ID_path_dict.keys()), "floorplans")
48
49
50  # In[4]:
51
52
53  list(ID_path_dict.items())[:10]
54
55
56  # In[5]:
57
58
59  with open(csv_to, "w", newline="", encoding="utf-8-sig") as csvfile:
60      listwriter = csv.writer(csvfile)
61      listwriter.writerow(["ID"])
62
63  IDs_error = []
64  for ID, path in tqdm(ID_path_dict.items(), desc="Processing plans"):
65      try:
66          listwriter.writerow([ID])
67      except:
68          IDs_error.append(ID)
69  print(len(IDs_error))
70  print(IDs_error)
71
72
73  # # analysis
74
75  # In[6]:
76
77
78  import pandas as pd
79
80  df = pd.read_csv(csv_to)
81  # df = df.set_index("ID")
82
83
84  # In[7]:
85
86
87  df

```

```
88
89
90 # In[8]:
91
92
93 df_old = pd.read_csv("processed_0812.csv")
94
95 df_old[~df_old.index.isin(df.index)]
96
```

## 17\_image\_size.py

```
1  #!/usr/bin/env python
2  # coding: utf-8
3
4  # In[1]:
5
6
7  get_ipython().run_line_magic("matplotlib", "inline")
8
9  from PIL import Image
10
11 import cv2, matplotlib
12 import numpy as np
13
14 from math import sqrt
15
16 import matplotlib.pyplot as plt
17
18 from os.path import expanduser, splitext
19 from os import scandir, makedirs
20
21 import csv
22
23 from tqdm.notebook import trange, tqdm
24
25 from pathlib import Path
26
27 debug = False # plot every steps
28
29
30 # In[2]:
31
32
33 from floorplan_analysis import read_from_csv
34
35
36 # # CSV
37
38 # In[3]:
39
40
41 dir_from = "/data/fp_img_processed/"
42
43 csv_to = "size.csv"
```

```

44
45  ### all of the plans
46  ID_path_dict = {splitext(f.name)[0]: f.path for f in scandir(dir_from) if f.is_file()}
47  print(len(ID_path_dict.keys()), "floorplans")
48
49
50  # In[4]:
51
52
53  list(ID_path_dict.items())[:10]
54
55
56  # In[5]:
57
58
59  with open(csv_to, "w", newline="", encoding="utf-8-sig") as csvfile:
60      listwriter = csv.writer(csvfile)
61
62      IDs_error = []
63      for ID, path in tqdm(ID_path_dict.items(), desc="Processing plans"):
64          try:
65              with Image.open(path) as im:
66                  width, height = im.size
67                  listwriter.writerow([ID, width, height])
68          except:
69              IDs_error.append(ID)
70      print(len(IDs_error))
71      print(IDs_error)
72
73
74  # # analysis
75
76  # In[6]:
77
78
79  import pandas as pd
80
81  df = pd.read_csv(csv_to, names=["ID", "width", "height"])
82  df = df.set_index("ID")
83
84
85  # In[7]:
86
87

```

```
88 df
89
90
91 # In[8]:
92
93
94 fig, axs = plt.subplots(1, 2, sharey=True, figsize=(11, 5), dpi=300)
95
96 df.width.plot.kde(ax=axs[0])
97 axs[0].set_title("Width of unit plans (m)")
98
99 df.height.plot.kde(ax=axs[1])
100 axs[1].set_title("Depth of unit plans (m)")
101
102 for ax in axs:
103     ax.set_xlim(0, 120)
104     ax.set_xticks(range(0, 120 + 1, 20))
105     ax.set_xticklabels(range(0, 24 + 1, 4))
106     ax.set_ylim(0)
107     ax.set_yticks([])
108
109 plt.tight_layout()
110 fig.savefig("fp_size_kde.pdf", bbox_inches="tight", pad_inches=0)
111
```

## 24\_dataset\_all\_train\_test\_56.py

```
1  #!/usr/bin/env python
2  # coding: utf-8
3
4  # In[1]:
5
6
7  # !pip install --upgrade pip
8  # !pip install pandas
9  # !cp ./fp_refined.csv /data
10
11
12 # In[2]:
13
14
15 get_ipython().run_line_magic('matplotlib', 'inline')
16
17 import numpy as np
18 import tensorflow as tf
19 import cv2
20 import matplotlib.pyplot as plt
21
22
23 import pandas as pd
24
25 from tqdm.notebook import trange, tqdm
26
27 # tf.enable_eager_execution() # tf2
28
29
30 # In[3]:
31
32
33 path_csv = "fp_refined.csv"
34 df = pd.read_csv(path_csv)
35 df
36
37
38 # In[4]:
39
40
41 df_image = pd.read_csv("processed.csv")
42 df_image
43
```

```

44
45 # In[5]:
46
47
48 df = df[df.id_after.isin(df_image.ID)]
49 df
50
51
52 # In[6]:
53
54
55 def read_mono_from_image_unicode(path):
56     """workaround for non-ascii filenames"""
57
58     stream = open(path, "rb")
59     bytes = bytearray(stream.read())
60     numpyarray = np.asarray(bytes, dtype=np.uint8)
61     mono = cv2.imdecode(numpyarray, cv2.IMREAD_UNCHANGED)
62
63     return mono
64
65
66 def fp_float_from_mono(mono):
67     """create 0 or 1 binary mask image from
68     wall/entrance/LDK/bedroom/balcony/bathroom stacked array"""
69
70     # AREA_WALL = 64
71     # AREA_ENTRANCE = 32
72     # AREA_LDK = 16
73     # AREA_BEDROOM = 8
74     # AREA_BALCONY = 4
75     # AREA_BATHROOM = 2
76
77     mask_bits = np.array([64, 32, 16, 8, 4, 2], dtype=np.uint8)
78     mask = np.broadcast_to(mask_bits, (*mono.shape[:2], 6))
79
80     unit_comb = (((np.expand_dims(mono, 2) & mask) > 0)).astype(np.float)
81
82     return unit_comb
83
84
85 def pad_fp(fp, width=112, height=112):
86     """place the fp at the bottom center of padded image."""
87     h, w = np.subtract(fp.shape[:2], (height, width))

```

```

88     if h > 0:
89         fp = fp[h : h + height, :, :]
90     if w > 0:
91         fp = fp[:, w // 2 : w // 2 + width, :]
92
93     h, w = np.subtract((height, width), fp.shape[:2])
94     fp = np.pad(fp, ((max(h, 0), 0), (max(w // 2, 0), max(w - w // 2, 0)), (0, 0)))
95     return fp
96
97
98 def visualize_fp(fps):
99     # adjusted for different luminance
100    channel_to_rgba = np.array(
101        [
102            [0.0, 0.0, 0.0, 0.0], # wall to black L0
103            [0.0, 0.33, 0.0, 0.0], # entrance to green L30
104            [1.0, 0.25, 0.0, 0.0], # LDK to red L57
105            [0.83, 0.87, 0.0, 0.0], # bedroom to yellow L85
106            [0.0, 0.26, 1.0, 0.0], # balcony to blue L40
107            [0.0, 0.81, 0.76, 0.0], # bathroom to cyan L75
108        ]
109    )
110
111    # make colors subtractive
112    channel_to_rgba[:, 0:3] -= 1
113
114    # put it on white
115    fps_rgba = np.clip(
116        np.array([1.0, 1.0, 1.0, 1.0]) + (np.array(fps) @ channel_to_rgba), 0, 1
117    )
118    return fps_rgba
119
120
121 # def _fp_from_string(bytes):
122 #     return np.frombuffer(bytes).reshape(28,28,6)
123
124
125 # In[7]:
126
127
128 # from os import makedirs
129 # makedirs('vis',exist_ok=True)
130
131

```

```

132 # 10001_57B
133 # 2112_49_0
134 id = "2112_49_0"
135 mono = read_mono_from_image_unicode(f"/data/fp_img_processed/{id}.png")
136 fp = fp_float_from_mono(mono)
137 fp = pad_fp(fp)
138
139 # fig = plt.figure(figsize=(5, 5), dpi=300)
140 plt.imshow(visualize_fp(fp), )
141 # plt.axis('off')
142
143 plt.tight_layout()
144 # fig.savefig(f"vis/{id}.pdf", bbox_inches="tight", pad_inches=0)
145
146
147 # In[8]:
148
149
150 def preprocess_fp(path):
151     mono = read_mono_from_image_unicode(path)
152     fp = fp_float_from_mono(mono)
153     fp = pad_fp(fp, 56, 56)
154     return fp
155
156
157 # In[9]:
158
159
160 def preprocess_year(year):
161     return max(0, (year - 1970) // 5)
162
163
164 # In[10]:
165
166
167 df_tfrecord = df.loc[
168     :, [
169         "Path",
170         "id_after",
171         "year",
172         "sido_cluster_code",
173         "norm_log_area",
174         "Rooms",

```

```

176         "Baths",
177     ],
178 ]
179
180
181 # In[11]:
182
183
184 df.APT_ID.unique().shape
185
186
187 # In[12]:
188
189
190 np.random.seed(1106)
191 id_test = np.random.choice(df.APT_ID.unique(), 3000)
192 id_test[:100]
193
194
195 # In[13]:
196
197
198 df_tfrecord[df_tfrecord.id_after.isin(df[~df.APT_ID.isin(id_test)].id_after)]
199
200
201 # In[14]:
202
203
204 rows_all = df_tfrecord.values
205 rows_train = df_tfrecord[
206     df_tfrecord.id_after.isin(df[~df.APT_ID.isin(id_test)].id_after)
207 ].values
208 rows_test = df_tfrecord[
209     df_tfrecord.id_after.isin(df[df.APT_ID.isin(id_test)].id_after)
210 ].values
211
212
213 # In[15]:
214
215
216 # The following functions can be used to convert a value to a type compatible
217 # with tf.Example.
218 # https://www.tensorflow.org/tutorials/load_data/tf_records
219
```

```

220
221     def _bytes_feature(value):
222         """Returns a bytes_list from a string / byte."""
223         return tf.train.Feature(bytes_list=tf.train.BytesList(value=[value]))
224
225
226     def _float_feature(value):
227         """Returns a float_list from a float / double."""
228         return tf.train.Feature(float_list=tf.train.FloatList(value=[value]))
229
230
231     def _int64_feature(value):
232         """Returns an int64_list from a bool / enum / int / uint."""
233         return tf.train.Feature(int64_list=tf.train.Int64List(value=[value]))
234
235
236     def _floats_array_feature(value):
237         """Returns a float_list from a numpy array of floats / doubles."""
238         # https://stackoverflow.com/questions/47861084/how-to-store-numpy-arrays-as-tfrecord
239         return tf.train.Feature(float_list=tf.train.FloatList(value=value.reshape(-1)))
240
241
242     def _int64s_array_feature(value):
243         """Returns an int64_list from a numpy array of ints."""
244         return tf.train.Feature(int64_list=tf.train.Int64List(value=value.reshape(-1)))
245
246
247     # In[16]:
248
249
250     # tf.train.Feature(float_list=tf.train.FloatList(value=processed.flatten()))
251
252     # _floats_array_feature(fp)
253
254
255     # In[17]:
256
257
258     def serialize_example(row):
259         """
260             Creates a tf.Example message ready to be written to a file.
261         """
262
263         # Create a dictionary mapping the feature name to the tf.Example-compatible

```

```

264     # data type.
265
266     fp = preprocess_fp(row[0])
267     year = preprocess_year(row[2])
268     feature = {
269         "floorplan": _floats_array_feature(fp.reshape(-1)),
270         "plan_id": _bytes_feature(row[1].encode("utf-8")),
271         "year": _int64_feature(year),  # 0~9
272         "sido": _int64_feature(row[3]),  # 0~8
273         "norm_area": _float_feature(row[4]),
274         "num_rooms": _int64_feature(row[5]),  # 1~7
275         "num_baths": _int64_feature(row[6]),  # 1~5
276     }
277
278     # Create a Features message using tf.train.Example.
279     example_proto = tf.train.Example(features=tf.train.Features(feature=feature))
280
281     return example_proto.SerializeToString()
282
283
284 # In[18]:
285
286
287 example_proto = tf.train.Example.FromString(serialize_example(rows_train[0]))
288 # example_proto
289
290
291 # In[19]:
292
293
294 path_all_tfrecord = "fp56.tfrecord"
295 path_train_tfrecord = "fp56_train.tfrecord"
296 path_test_tfrecord = "fp56_test.tfrecord"
297
298 options_gzip = tf.io.TFRecordOptions(compression_type="GZIP")  # tf2
299
300 errors = []
301
302
303 for (path, rows) in zip(
304     [path_all_tfrecord, path_train_tfrecord, path_test_tfrecord],
305     [rows_all, rows_train, rows_test],
306 ):
307     with tf.io.TFRecordWriter(path=path, options=options_gzip) as writer:

```

```
308     for row in tqdm(rows, desc="Processing " + path):
309         try:
310             serialized_example = serialize_example(row)
311             writer.write(serialized_example)
312         except:
313             errors.append(row[1])
314
315
316     if errors:
317         print(errors)
318
```

## 41\_visualize\_floorplan.py

```
1  #!/usr/bin/env python
2  # coding: utf-8
3
4  # In[1]:
5
6
7  # !pip install --upgrade pip
8  # !pip install pandas
9  # !cp ./fp_refined.csv /data
10
11
12 # In[2]:
13
14
15 get_ipython().run_line_magic("matplotlib", "inline")
16
17 import numpy as np
18 import tensorflow as tf
19 import cv2
20 import matplotlib.pyplot as plt
21
22
23 import pandas as pd
24
25 from tqdm.notebook import trange, tqdm
26
27 # tf.enable_eager_execution() # tf2
28
29
30 # In[3]:
31
32
33 def read_mono_from_image_unicode(path):
34     """workaround for non-ascii filenames"""
35
36     stream = open(path, "rb")
37     bytes = bytearray(stream.read())
38     numpyarray = np.asarray(bytes, dtype=np.uint8)
39     mono = cv2.imdecode(numpyarray, cv2.IMREAD_UNCHANGED)
40
41     return mono
42
43
```

```

44     def fp_float_from_mono(mono):
45         """create 0 or 1 binary mask image from
46         wall/entrance/LDK/bedroom/balcony/bathroom stacked array"""
47
48         # AREA_WALL = 64
49         # AREA_ENTRANCE = 32
50         # AREA_LDK = 16
51         # AREA_BEDROOM = 8
52         # AREA_BALCONY = 4
53         # AREA_BATHROOM = 2
54
55         mask_bits = np.array([64, 32, 16, 8, 4, 2], dtype=np.uint8)
56         mask = np.broadcast_to(mask_bits, (*mono.shape[:2], 6))
57
58         unit_comb = (((np.expand_dims(mono, 2) & mask) > 0)).astype(np.float)
59
60         return unit_comb
61
62
63     def pad_fp(fp, width=112, height=112):
64         """place the fp at the bottom center of padded image."""
65         h, w = np.subtract(fp.shape[:2], (height, width))
66
67         if h > 0:
68             fp = fp[h : h + height, :, :]
69
70         if w > 0:
71             fp = fp[:, w // 2 : w // 2 + width, :]
72
73         h, w = np.subtract((height, width), fp.shape[:2])
74         fp = np.pad(fp, ((max(h, 0), 0), (max(w // 2, 0), max(w - w // 2, 0)), (0, 0)))
75
76         return fp
77
78
79     def visualize_fp(fps):
80         # adjusted for different luminance
81         channel_to_rgba = np.array(
82             [
83                 [0.0, 0.0, 0.0, 0.0], # wall to black L0
84                 [0.0, 0.33, 0.0, 0.0], # entrance to green L30
85                 [1.0, 0.25, 0.0, 0.0], # LDK to red L57
86                 [0.83, 0.87, 0.0, 0.0], # bedroom to yellow L85
87                 [0.0, 0.26, 1.0, 0.0], # balcony to blue L40
88                 [0.0, 0.81, 0.76, 0.0], # bathroom to cyan L75
89             ]
90         )

```

```

88
89     # make colors subtractive
90     channel_to_rgba[:, 0:3] -= 1
91
92     # put it on white
93     fps_rgba = np.clip(
94         np.array([1.0, 1.0, 1.0, 1.0]) + (np.array(fps) @ channel_to_rgba), 0, 1
95     )
96
97     return fps_rgba
98
99
100    # def _fp_from_string(bytes):
101    #     return np.frombuffer(bytes).reshape(28,28,6)
102
103    # In[21]:
104
105
106    from os import makedirs
107
108
109    # 10001_57B
110    # 2112_49_0
111
112
113    def visualize_from_file(ids, dir_to="vis"):
114
115        makedirs(dir_to, exist_ok=True)
116
117        for id in ids:
118            mono = read_mono_from_image_unicode(f"/data/fp_img_processed/{id}.png")
119            fp = fp_float_from_mono(mono)
120            fp_rgba = visualize_fp(fp)
121
122            fp_light = (
123                cv2.cvtColor(fp_rgba.astype(np.uint8) * 255, cv2.COLOR_RGB2Lab)[ :, :, 0]
124                / 100
125            )
126            fp_rgba = pad_fp(fp_rgba, fp_light.shape[1], fp_light.shape[0])
127
128            fig = plt.figure(figsize=(5, 5), dpi=300)
129            plt.imshow(fp_rgba)
130            plt.axis("off")
131

```

```

132         plt.tight_layout()
133         fig.savefig(f"{dir_to}/{id}.png", bbox_inches="tight", pad_inches=0)
134         plt.close()
135
136
137 visualize_from_file(["10001_57B"])
138
139
140 # In[22]:
141
142
143 from pathlib import Path
144 import csv
145
146
147 def read_from_csv(filepath, columns=False):
148     if Path(filepath).is_file():
149
150         with open(filepath, "r", newline="", encoding="utf-8-sig") as csvfile:
151             listreader = csv.reader(csvfile)
152             if columns:
153                 columns = next(listreader)
154             readlist = list(listreader)
155
156     else:
157         columns = []
158         readlist = []
159
160     if columns:
161         return columns, readlist
162     else:
163         return readlist
164
165
166 top_ids = read_from_csv("vgg_activation_top10.csv")
167 top_ids
168
169
170 # In[23]:
171
172
173 ids_flat = [id for line in top_ids for id in line]
174 ids_flat
175

```

```
176
177 # In[24]:
178
179
180 visualize_from_file(ids_flat)
```

## 61\_VGG\_CAM.py

```
1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # In[1]:
5
6
7 import tensorflow as tf
8 import tensorflow.keras.backend as K
9
10 from tensorflow.keras.models import Sequential, Model
11
12 from tensorflow.keras.layers import (
13     Input,
14     Dense,
15     Reshape,
16     Flatten,
17     Dropout,
18     BatchNormalization,
19     Activation,
20     ZeroPadding2D,
21     LeakyReLU,
22     UpSampling2D,
23     Conv2D,
24     Convolution2D,
25     MaxPooling2D,
26     Concatenate,
27     GaussianNoise,
28     GaussianDropout,
29     Lambda,
30     GlobalAveragePooling2D,
31 )
32
33 from tensorflow.keras.optimizers import Adam
34
35 from tensorflow.keras.utils import to_categorical
36
37 import h5py
38 import pickle
39
40 import matplotlib.pyplot as plt
41 import numpy as np
42 import pandas as pd
43
```

```

44 import os
45 import pathlib
46
47 import time
48
49 import math
50
51
52 # In[2]:
53
54
55 print("Tensorflow version: ", tf.version.VERSION) # tf2
56 print("Keras version: ", tf.keras.__version__) # 2.2.4-tf
57
58 # tf.enable_eager_execution() # tf2
59 print("Is eager execution enabled: ", tf.executing_eagerly())
60 print("Is there a GPU available: ", tf.test.is_gpu_available())
61
62
63 # In[3]:
64
65
66 path_train_tfrecord = "fp56_train.tfrecord"
67 path_test_tfrecord = "fp56_test.tfrecord"
68
69
70 # # model save dir
71
72 # In[4]:
73
74
75 dir_model = "vgg_cam/"
76 pathlib.Path(dir_model).mkdir(parents=True, exist_ok=True)
77
78
79 # In[5]:
80
81
82 fp_dim = (56, 56, 6)
83
84
85 def _parse_function(example_proto):
86     # Create a description of the features.
87     feature_description = {

```

```

88     "floorplan": tf.io.FixedLenFeature(
89         fp_dim, tf.float32, default_value=tf.zeros(fp_dim, tf.float32)
90     ),
91     "plan_id": tf.io.FixedLenFeature([], tf.string, default_value=""),
92     "year": tf.io.FixedLenFeature([], tf.int64, default_value=-1), # 0~9
93     # "sido": tf.FixedLenFeature([], tf.int64, default_value=-1),
94     # "norm_area": tf.FixedLenFeature([], tf.float32, default_value=0.0),
95     # "num_rooms": tf.FixedLenFeature([], tf.int64, default_value=-1),
96     # "num_baths": tf.FixedLenFeature([], tf.int64, default_value=-1),
97 }
98
99 # Parse the input tf.Example proto using the dictionary above.
100 parsed_example = tf.io.parse_single_example(example_proto, feature_description)
101
102 return parsed_example["floorplan"], parsed_example["year"]
103
104 # In[6]:
105
106
107
108 def _onehot_year(fp, year):
109     year_onehot = tf.one_hot(year, 10) # 1970~4 -> 0, 2015~9 -> 9
110     return (fp, year_onehot)
111
112
113 # In[7]:
114
115
116 def create_dataset(filepath):
117     # This works with arrays as well
118     dataset = tf.data.TFRecordDataset(filepath, compression_type="GZIP")
119
120     # Maps the parser on every filepath in the array. You can set the number of parallel loaders here
121     dataset = dataset.map(_parse_function, num_parallel_calls=4)
122
123     ### preprocess the features
124
125     # won't use it. use sparse_categorical_crossentropy instead of categorical_crossentropy.
126     #     dataset = dataset.map(_onehot_year, num_parallel_calls=4)
127
128     return dataset
129
130
131 # In[8]:

```

```

132
133
134 def VGG16_convolutions():
135     if K.image_data_format() == "channels_last":
136         input_shape = (fp_dim[0], fp_dim[1], fp_dim[2])
137     else:
138         input_shape = (fp_dim[2], fp_dim[0], fp_dim[1])
139
140     model = Sequential()
141     model.add(GaussianNoise(0.1, input_shape=input_shape))
142
143     model.add(Conv2D(64, (3, 3), activation="relu", name="conv1_1", padding="same"))
144     model.add(Conv2D(64, (3, 3), activation="relu", name="conv1_2", padding="same"))
145     model.add(MaxPooling2D((2, 2), strides=(1, 1), padding="same"))
146
147     model.add(Conv2D(128, (3, 3), activation="relu", name="conv2_1", padding="same"))
148     model.add(Conv2D(128, (3, 3), activation="relu", name="conv2_2", padding="same"))
149     model.add(MaxPooling2D((2, 2), strides=(1, 1), padding="same"))
150
151     model.add(Conv2D(256, (3, 3), activation="relu", name="conv3_1", padding="same"))
152     model.add(Conv2D(256, (3, 3), activation="relu", name="conv3_2", padding="same"))
153     model.add(Conv2D(256, (3, 3), activation="relu", name="conv3_3", padding="same"))
154     model.add(MaxPooling2D((2, 2), strides=(2, 2)))
155
156     model.add(Conv2D(512, (3, 3), activation="relu", name="conv4_1", padding="same"))
157     model.add(Conv2D(512, (3, 3), activation="relu", name="conv4_2", padding="same"))
158     model.add(Conv2D(512, (3, 3), activation="relu", name="conv4_3", padding="same"))
159     model.add(MaxPooling2D((2, 2), strides=(2, 2)))
160
161     model.add(Conv2D(512, (3, 3), activation="relu", name="conv5_1", padding="same"))
162     model.add(Conv2D(512, (3, 3), activation="relu", name="conv5_2", padding="same"))
163     model.add(Conv2D(512, (3, 3), activation="relu", name="conv5_3", padding="same"))
164
165     return model
166
167 # In[9]:
168
169
170 num_classes = 10
171
172
173 def create_model():
174     model = VGG16_convolutions()
175

```

```

176     model.add(GlobalAveragePooling2D())
177     model.add(Dropout(0.5))
178     model.add(Dense(num_classes, activation="softmax"))
179
180     model.compile(
181         optimizer="sgd", loss="sparse_categorical_crossentropy", metrics=["accuracy"])
182     )
183     return model
184
185
186 # In[10]:
187
188
189 callback_checkpoint = tf.keras.callbacks.ModelCheckpoint(
190     dir_model + "model-{epoch:02d}-{val_loss:.2f}-{val_accuracy:.1%}.hdf5",
191     save_weights_only=True,
192     verbose=1,
193 )
194
195
196 # In[11]:
197
198
199 callback_stop = tf.keras.callbacks.EarlyStopping(monitor="val_loss", patience=20)
200
201
202 # # run
203
204 # In[12]:
205
206
207 # .repeat().shuffle(4096).batch(8)
208
209 train_dataset = create_dataset(path_train_tfrecord).shuffle(1024).batch(8)
210 test_dataset = create_dataset(path_test_tfrecord).batch(8)
211
212 train_dataset, test_dataset
213
214
215 # In[13]:
216
217
218 model = create_model()
219 model.summary()

```

```

220
221
222 # In[14]:
223
224
225 path_best = dir_model + "model-15-1.36.hdf5"
226 epoch_best = 0 # 0 if starting from fresh
227
228 if epoch_best and os.path.exists(path_best):
229     model.load_weights(path_best)
230     history = model.fit(
231         train_dataset,
232         epochs=50,
233         initial_epoch=epoch_best,
234         validation_data=test_dataset,
235         callbacks=[callback_checkpoint, callback_stop],
236     )
237 else:
238     history = model.fit(
239         train_dataset,
240         epochs=50,
241         validation_data=test_dataset,
242         callbacks=[callback_checkpoint, callback_stop],
243     )
244
245
246 # In[15]:
247
248
249 history.history
250
251
252 # In[16]:
253
254
255 df_hist = pd.DataFrame(
256     history.history,
257     index=range(epoch_best + 1, epoch_best + len(history.history["loss"]) + 1),
258 )
259 df_hist.index.name = "epoch"
260
261
262 # In[17]:
263
```

```
264
265 df_hist
266
267
268 # In[18]:
269
270
271 path_hist = dir_model + "history.csv"
272 df_hist.to_csv(path_hist, mode="a", header=not os.path.exists(path_hist))
```

## 62\_VGG\_CAM\_prediction.py

```
1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # !pip install --upgrade pip matplotlib
5
6 # In[1]:
7
8
9 import matplotlib.pyplot as plt
10 import numpy as np
11 import pandas as pd
12
13
14 # In[2]:
15
16
17 path_all_tfrecord = "fp56.tfrecord"
18
19 # path_train_tfrecord = "fp56_train.tfrecord"
20 # path_test_tfrecord = "fp56_test.tfrecord"
21
22
23 # In[3]:
24
25
26 dir_model = "vgg_cam/"
27 path_best = dir_model + "model-17-1.17-53.3%.hdf5"
28 path_best
29
30
31 # In[4]:
32
33
34 from fp_tensorflow import _parse_pair_56, _parse_single_56
35 from fp_tensorflow import create_pair_56_dataset, create_single_dataset
36 from fp_tensorflow import VGG16_convolutions
37 from fp_tensorflow import create_vgg_5y_model
38
39 all_dataset = create_pair_56_dataset(path_all_tfrecord, "floorplan", "year").batch(64)
40
41 model = create_vgg_5y_model()
42 model.load_weights(path_best)
43
```

```

44
45 # # predict
46
47 # In[5]:
48
49
50 predictions = model.predict(all_dataset, verbose=1)
51
52
53 # In[6]:
54
55
56 predictions = np.argmax(predictions, 1)
57
58
59 # In[7]:
60
61
62 predictions[:128]
63
64
65 # In[8]:
66
67
68 all_year = create_single_dataset(path_all_tfrecord, "year")
69 year_true = np.fromiter((y.numpy() for y in all_year), int)
70 year_true.shape, year_true[:10]
71
72
73 # In[9]:
74
75
76 all_id = create_single_dataset(path_all_tfrecord, "plan_id")
77 ids = [i.numpy().decode() for i in all_id]
78 ids[:10]
79
80
81 # In[10]:
82
83
84 df = pd.DataFrame(
85     zip(ids, year_true, predictions), columns=["ID", "true", "prediction"]
86 )
87 df

```

```

88
89
90 # In[11]:
91
92
93 df.to_csv("vgg_5y_prediction.csv")
94
95
96 # In[12]:
97
98
99 crosstab = pd.crosstab(df.true, df.prediction)
100 crosstab[0] = 0
101 crosstab = crosstab.reindex(index=range(10), columns=range(10), fill_value=0)
102 crosstab
103
104
105 # In[13]:
106
107
108 fig = plt.figure(figsize=(7, 5), dpi=300)
109 ax = fig.gca()
110
111 c = ax.pcolor(crosstab.transpose(), cmap="BuGn")
112
113 ax.set_aspect("equal")
114 ax.set_xlabel("True Completion Year")
115 ax.set_ylabel("Prediction")
116
117 ax.set_xticks(range(0, 11, 2))
118 ax.set_yticks(range(0, 11, 2))
119 ax.set_xticklabels(range(1970, 2021, 10))
120 ax.set_yticklabels(range(1970, 2021, 10))
121
122 fig.colorbar(c, ax=ax)
123
124 fig.savefig("vgg_5y_heatmap.pdf", bbox_inches="tight", pad_inches=0)
125 fig.savefig("vgg_5y_heatmap.png", bbox_inches="tight", pad_inches=0)
126
127
128 # ### 눈 먼 정확도는 딱 한 구간 틀린 게 많다는 걸 보여주지 못함...
129
130 # In[14]:
131

```

```
132
133     correct = df[df.true == df.prediction].shape[0]
134     total = df.shape[0]
135     correct, total, f'{correct/total:.2%}'
136
137
138     # In[15]:
139
140
141     one_off = (
142         df[df.true - 1 == df.prediction].shape[0]
143         + df[df.true + 1 == df.prediction].shape[0]
144     )
145     one_off, f'{one_off/total:.2%}', f'{{(correct+one_off)}/{total:.2%}}'
```

## 63\_VGG\_CAM\_activation.py

```
1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # In[28]:
5
6
7 # import tensorflow as tf
8 # import tensorflow.keras.backend as K
9
10 from tensorflow.keras.models import Sequential, Model
11
12 # from tensorflow.keras.layers import (
13 #     Input,
14 #     Dense,
15 #     Reshape,
16 #     Flatten,
17 #     Dropout,
18 #     BatchNormalization,
19 #     Activation,
20 #     ZeroPadding2D,
21 #     LeakyReLU,
22 #     UpSampling2D,
23 #     Conv2D,
24 #     Convolution2D,
25 #     MaxPooling2D,
26 #     Concatenate,
27 #     GaussianNoise,
28 #     GaussianDropout,
29 #     Lambda,
30 #     GlobalAveragePooling2D,
31 # )
32
33 # from tensorflow.keras.optimizers import Adam
34
35 # from tensorflow.keras.utils import to_categorical
36
37 # import h5py
38 # import pickle
39 # import csv
40
41 import matplotlib.pyplot as plt
42 from matplotlib.colors import DivergingNorm
43
```

```
44 import numpy as np
45 import pandas as pd
46
47 # import os
48 # import pathlib
49 # from pathlib import Path
50
51 # import time
52
53 # import math
54
55
56 # In[2]:
57
58
59 path_all_tfrecord = "fp56.tfrecord"
60
61 # path_train_tfrecord = "fp56_train.tfrecord"
62 # path_test_tfrecord = "fp56_test.tfrecord"
63
64
65 # In[3]:
66
67
68 dir_model = "vgg_cam/"
69 path_best = dir_model + "model-17-1.17-53.3%.hdf5"
70 path_best
71
72
73 # # load
74
75 # In[4]:
76
77
78 df = pd.read_csv("vgg_5y_prediction.csv", index_col=0)
79 df
80
81
82 # In[5]:
83
84
85 predictions = df.prediction.to_numpy()
86 predictions
87
```

```

88
89 # In[6]:
90
91
92 year_true = df.true.to_numpy()
93 year_true
94
95
96 # In[7]:
97
98
99 ids = df.ID.tolist()
100 ids[60:70]
101
102
103 # # run
104
105 # In[9]:
106
107
108 from fp_tensorflow import _parse_pair_56, _parse_single_56
109 from fp_tensorflow import create_pair_56_dataset, create_single_dataset
110 from fp_tensorflow import VGG16_convolutions
111 from fp_tensorflow import create_vgg_5y_model
112
113 all_dataset = create_pair_56_dataset(path_all_tfrecord, "floorplan", "year").batch(64)
114
115 model = create_vgg_5y_model()
116 model.load_weights(path_best)
117
118
119 # In[10]:
120
121
122 model.summary()
123
124
125 # In[11]:
126
127
128 # Get the 512 input weights to the softmax.
129 class_weights = model.layers[-1].get_weights()[0]
130
131

```

```
132 # In[12]:  
133  
134  
135 class_weights.shape  
136  
137  
138 # In[13]:  
139  
140  
141 class_weights.mean(), class_weights.std()  
142  
143  
144 # In[26]:  
145  
146  
147 pd.Series(class_weights.mean(axis=1)).plot.kde()  
148  
149  
150 # In[23]:  
151  
152  
153 # bias for 10 of 5-year classes  
154 # model.layers[-1].get_weights()[1]  
155  
156  
157 # # global average pool output  
158  
159 # In[29]:  
160  
161  
162 gap_model = Model(inputs=model.input, outputs=model.layers[-2].output)  
163  
164  
165 # In[30]:  
166  
167  
168 gap_outputs = gap_model.predict(all_dataset, verbose=1)  
169  
170  
171 # In[32]:  
172  
173  
174 gap_outputs.shape, gap_outputs.mean(), gap_outputs.std()  
175
```

```
176
177 # In[41]:
178
179 pd.Series(gap_outputs[116]).plot.kde(bw_method=0.02)
180
181
182
183 # In[42]:
184
185
186 df_gap = pd.DataFrame(gap_outputs)
187 df_gap
188
189
190 # In[43]:
191
192
193 df_gap.to_csv("vgg_5y_activation.csv.gz")
194
```

## 65\_biclustering.py

```
1  #!/usr/bin/env python
2  # coding: utf-8
3
4  # In[1]:
5
6
7  import pandas as pd
8  import matplotlib.pyplot as plt
9  import numpy as np
10
11
12 # In[2]:
13
14
15 df = pd.read_csv("vgg_5y_activation.csv.gz", index_col=0)
16 df
17
18 # 신경망 출력값의 범위
19
20
21 # In[3]:
22
23
24 flatten = pd.DataFrame(df.to_numpy().reshape(-1))
25 flatten.sample(frac=0.01, random_state=1106).plot.kde()
26
27
28 # In[4]:
29
30
31 flatten.describe()
32
33
34 # 바이클러스터링 군집 수 결정
35
36 # In[5]:
37
38
39 from sklearn.cluster import MiniBatchKMeans
40
41
42 # In[6]:
43
```

```

44
45  clust = MiniBatchKMeans(
46      n_clusters=30, batch_size=100, verbose=0, random_state=1106,
47  ).fit(df)
48
49  clust.inertia_
50
51  # In[7]:
52
53
54
55  inertia_dict = {
56      i: MiniBatchKMeans(n_clusters=i, random_state=1106).fit(df).inertia_
57      for i in range(2, 31)
58  }
59
60
61  # In[8]:
62
63
64  inertia_dict
65
66
67  # In[293]:
68
69
70  fig = plt.figure(figsize=(10, 3), dpi=300)
71
72  ax = pd.Series(inertia_dict).plot()
73  ax.set_xlim(0, 30)
74  ax.set_ylim(0)
75
76  ax2 = ax.twinx()
77  (pd.Series(inertia_dict).diff() * (-1)).plot(c="tab:red", ax=ax2)
78  # ax2.set_ylim(0)
79  ax2.axhline(0, c="tab:red", ls=":")
80  ax2.tick_params(axis="y", labelcolor="tab:red")
81
82  n_clusters = 16  # -----
83
84  [ax.axvline(n + 0.5, c="k", ls="--") for n in [n_clusters]]
85  ax.set_xticks(list(range(0, 30 + 1, 10)) + [n_clusters])
86
87  plt.show()

```

```

88 fig.savefig("biclust_n.png", bbox_inches="tight", pad_inches=0)
89 fig.savefig("biclust_n.pdf", bbox_inches="tight", pad_inches=0)
90
91
92 #
93
94 # In[12]:
95
96
97 import matplotlib.colors as colors
98
99 fig = plt.figure(figsize=(5, 7), dpi=300)
100 ax = fig.gca()
101
102 im = ax.matshow(df[::100], vmin=0, vmax=1, cmap="viridis",)
103
104 ax.set_yticks(range(0, 500 + 1, 100))
105 ax.set_yticklabels(["f'{t}:'" for t in range(0, 50000 + 1, 10000)])
106
107 fig.colorbar(im, shrink=0.5)
108
109 plt.show()
110 fig.savefig("biclust_before.png", bbox_inches="tight", pad_inches=0)
111 fig.savefig("biclust_before.pdf", bbox_inches="tight", pad_inches=0)
112
113
114 # In[13]:
115
116
117 df_refined = pd.read_csv("fp_refined.csv")
118 df_refined
119
120
121 # In[14]:
122
123
124 df_vgg = pd.read_csv("vgg_5y_prediction.csv")
125
126
127 # In[294]:
128
129
130 df_clust = df_vgg.join(
131     df_refined.set_index("id_after")[

```

```

132         ["year", "sido_cluster_code", "Area", "Rooms", "Baths"]
133     ],
134     on="ID",
135 )
136 df_clust
137
138
139 # In[295]:
140
141
142 n_clusters = 16
143
144
145 # In[296]:
146
147
148 from sklearn.cluster import SpectralCoclustering
149
150 model = SpectralCoclustering(n_clusters=n_clusters, random_state=1106, n_jobs=6).fit(df)
151
152 df_clust["cluster"] = model.row_labels_
153 order = df_clust.groupby("cluster")["year"].mean().argsort().values
154 rank = order.argsort()
155 df_clust["cluster"] = rank[df_clust.cluster]
156
157 df_clust.groupby("cluster").year.mean()
158
159
160 # In[297]:
161
162
163 fig = plt.figure(figsize=(10, 3), dpi=300)
164
165 axs = df_clust.groupby("cluster")["year"].plot.kde(bw_method=0.5)
166 axs[0].set_xlim(1969, 2020)
167 axs[0].set_ylim(0)
168
169 # plt.legend()
170
171
172 # In[300]:
173
174
175 ordered = df.iloc[

```

```

176     np.argsort(rank[model.row_labels_]), np.argsort(rank[model.column_labels_])
177 ]
178
179 fig = plt.figure(figsize=(5, 7), dpi=300)
180 ax = fig.gca()
181
182 im = ax.matshow(ordered[::-100], vmin=0, vmax=1, cmap="viridis")
183
184 ax.set_yticks(range(0, 500 + 1, 100))
185 ax.set_yticklabels([f"{t:}" for t in range(0, 50000 + 1, 10000)])
186
187 fig.colorbar(im, shrink=0.5)
188
189 plt.show()
190 fig.savefig("biclust_after.png", bbox_inches="tight", pad_inches=0)
191 fig.savefig("biclust_after.pdf", bbox_inches="tight", pad_inches=0)
192
193
194 # In[301]:
195
196
197 df_mean = (
198     df.groupby(df_clust[["cluster"]])
199     .mean()
200     .groupby(rank[model.column_labels_], axis=1)
201     .mean()
202 )
203
204 fig = plt.figure(figsize=(5, 5), dpi=300)
205 ax = fig.gca()
206
207 im = ax.matshow(df_mean)
208 fig.colorbar(im)
209
210 ax.set_xticks(range(n_clusters))
211 ax.set_yticks(range(n_clusters))
212
213 ax.set_xticklabels(range(1, n_clusters + 1))
214 ax.set_yticklabels(range(1, n_clusters + 1))
215
216 plt.show()
217 fig.savefig("biclust_mean.png", bbox_inches="tight", pad_inches=0)
218 fig.savefig("biclust_mean.pdf", bbox_inches="tight", pad_inches=0)
219

```

```

220
221 # In[465]:
222
223
224 from matplotlib.colors import DivergingNorm
225
226 df_corr = df.groupby(rank[model.column_labels_], axis=1).sum().corr()
227
228 # Generate a mask for the upper triangle
229 mask = np.triu(np.ones_like(df_corr, dtype=np.bool))
230
231
232 fig = plt.figure(figsize=(5, 5), dpi=300)
233 ax = fig.gca()
234 im = ax.matshow(
235     np.ma.array(df_corr, mask=mask), cmap="coolwarm", norm=DivergingNorm(0),
236 )
237
238 fig.colorbar(im)
239
240 # Hide the right and top spines
241 ax.spines["right"].set_visible(False)
242 ax.spines["top"].set_visible(False)
243
244 ax.xaxis.tick_bottom()
245
246 ax.set_xticks(range(n_clusters))
247 ax.set_yticks(range(n_clusters))
248
249 ax.set_xticklabels(range(1, n_clusters + 1))
250 ax.set_yticklabels(range(1, n_clusters + 1))
251
252 plt.show()
253 fig.savefig("biclust_corr.png", bbox_inches="tight", pad_inches=0)
254 fig.savefig("biclust_corr.pdf", bbox_inches="tight", pad_inches=0)
255
256
257 # In[488]:
258
259
260 plt.matshow(np.ma.array((df_corr >= 0.5), mask=mask))
261
262 ax = plt.gca()
263 ax.set_xticks(range(n_clusters))

```

```

264     ax.set_yticks(range(n_clusters))
265
266     ax.set_xticklabels(range(1, n_clusters + 1))
267     ax.set_yticklabels(range(1, n_clusters + 1))
268
269     plt.show()
270
271
272     # 1번부터 16번까지의 평면 계획 특성 요인은 각 요인과 짹지어져 군집화된 평면들의 평균 준공연도를 기준으로 정렬한 것이다.
273     # 인접한? 요인들은 서로 비슷한 시기의 평면 계획 특성을 나타낸다.
274     # 각 시기에 해당하는 평면에서는 해당 시기의 여러 계획 특성이 함께 나타나기 때문에,
275     # 서로 인접한 평면 계획 특성 요인들끼리는 양의 상관관계를 보인다.
276     #
277     # 5와 12, 6과 15는 서로 다른 시기의 평면에서 나타나는 계획 특성 요인이지만 양의 상관관계를 보인다.
278     # 대부분의 요인이 서로 상관관계가 없거나 음의 상관관계를 보이는 것과 다르다.
279     # 해당 평면 군집 사이에 계획적 공통점이 있다는 점을 시사한다.
280
281     # In[498]:
282
283
284     plt.matshow(np.ma.array(df_corr <= -0.3, mask=mask),)
285
286     ax = plt.gca()
287     ax.set_xticks(range(n_clusters))
288     ax.set_yticks(range(n_clusters))
289
290     ax.set_xticklabels(range(1, n_clusters + 1))
291     ax.set_yticklabels(range(1, n_clusters + 1))
292
293     plt.show()
294
295
296     # 1990년대 평면을 포함하는 1~7번 요인과
297     # 2010년대 평면을 포함하는 13~16번 요인 사이에서는
298     # 대다수에서 역의 상관관계가 두드러진다.
299     # 딥 러닝 모형이 학습한 1990년대와 2010년대의 평면 계획 특성은
300     # 서로 다른 시기의 계획 특성을 잘 구분할 수 있다.
301     #
302     # 한편,
303     # 3, 6, 15번 요인은
304     # 2000년대 평면을 포함하는 8~12번 요인과 역의 상관관계를 보인다.
305     # 2000년대 평면에서 잘 나타나지 않았던 계획 경향을 나타낸다.
306
307     # In[305]:

```

```

308
309
310 rank[model.column_labels_]
311
312
313 # In[306]:
314
315
316 np.savetxt("biclust_col.txt", rank[model.column_labels_], "%.\u00d7")
317
318
319 # In[307]:
320
321
322 df_clust.to_csv("biclust.csv")
323
324
325 # In[308]:
326
327
328 pd.crosstab(df_clust.cluster, df_clust.Rooms)
329
330
331 # In[309]:
332
333
334 pd.crosstab(df_clust.cluster, df_clust.Rooms, normalize="index")
335
336
337 # In[360]:
338
339
340 from matplotlib.ticker import PercentFormatter
341
342 fig = plt.figure(figsize=(5, 5), dpi=300)
343 ax = fig.gca()
344
345 pd.crosstab(df_clust.cluster, df_clust.Rooms, normalize="index").plot.bar(
346     stacked=True, ax=ax
347 ).legend(loc="center left", bbox_to_anchor=(1.0, 0.5))
348
349 ax.set_xticks(range(n_clusters))
350 ax.set_xticklabels(range(1, n_clusters + 1))
351
```

```

352     ax.set_ylim(0, 1)
353     ax.yaxis.set_major_formatter(PercentFormatter(1))
354     fig.savefig("biclust_rooms.png", bbox_inches="tight", pad_inches=0)
355     fig.savefig("biclust_rooms.pdf", bbox_inches="tight", pad_inches=0)
356
357
358     # 대부분에서 침실 3개가 가장 많고 그 다음으로 침실 4개가 2위인 것이 일반적.
359     #
360     # 13번은 침실 1개가 가장 많음.
361     #
362     # 0, 2번은 침실 3개 다음으로 침실 2개가 뒤따름. (2위)
363
364     # In[361]:
365
366
367     from matplotlib.ticker import PercentFormatter
368
369     fig = plt.figure(figsize=(5, 5), dpi=300)
370     ax = fig.gca()
371
372     pd.crosstab(df_clust.cluster, df_clust.Baths, normalize="index").plot.bar(
373         stacked=True, ax=ax
374     ).legend(loc="center left", bbox_to_anchor=(1.0, 0.5))
375
376     ax.set_xticks(range(n_clusters))
377     ax.set_xticklabels(range(1, n_clusters + 1))
378
379     ax.set_ylim(0, 1)
380     ax.yaxis.set_major_formatter(PercentFormatter(1))
381     fig.savefig("biclust_baths.png", bbox_inches="tight", pad_inches=0)
382     fig.savefig("biclust_baths.pdf", bbox_inches="tight", pad_inches=0)
383
384
385     # 대체로 화장실 2개가 가장 많지만,
386     # 침실 1개가 가장 많은 13번과
387     # 침실 2개가 2위인 0, 2번은
388     # 화장실 1개가 가장 많음.
389
390     # In[312]:
391
392
393     cmap = plt.get_cmap("tab20")
394     colors = cmap(np.linspace(0, 0.5 - 1 / cmap.N, cmap.N // 2))
395     colors

```

```

396
397
398 # In[313]:
399
400
401 from matplotlib.colors import LinearSegmentedColormap
402
403 tab20half = LinearSegmentedColormap.from_list("tab20_Lower_Half", colors)
404
405
406 # In[362]:
407
408
409 from matplotlib.ticker import PercentFormatter
410
411 fig = plt.figure(figsize=(5, 5), dpi=300)
412 ax = fig.gca()
413
414 pd.crosstab(df_clust.cluster, df_clust.true, normalize="index").plot.bar(
415     stacked=True, cmap=tab20half, ax=ax
416 ).legend(
417     [
418         "1969-74",
419         "1975-79",
420         "1980-84",
421         "1985-89",
422         "1990-94",
423         "1995-99",
424         "2000-04",
425         "2005-09",
426         "2010-14",
427         "2015-19",
428     ],
429     loc="center left",
430     bbox_to_anchor=(1.0, 0.5),
431 )
432
433 ax.set_xticks(range(n_clusters))
434 ax.set_xticklabels(range(1, n_clusters + 1))
435
436 ax.set_xlim(0, 1)
437 ax.yaxis.set_major_formatter(PercentFormatter(1))
438 fig.savefig("biclust_year.png", bbox_inches="tight", pad_inches=0)
439 fig.savefig("biclust_year.pdf", bbox_inches="tight", pad_inches=0)

```

```

440
441
442 # 각 군집이 서로 다른 시기에 따라 잘 묶임
443
444 # In[315]:
445
446
447 tab20_sido = LinearSegmentedColormap.from_list(
448     "9 colors from tab20",
449     plt.get_cmap("tab20")([0, 0.05, 0.10, 0.15, 0.20, 0.30, 0.35, 0.40, 0.50]),
450 )
451
452
453 # In[363]:
454
455
456 from matplotlib.ticker import PercentFormatter
457
458 fig = plt.figure(figsize=(5, 5), dpi=300)
459 ax = fig.gca()
460
461 pd.crosstab(df_clust.cluster, df_clust.sido_cluster_code, normalize="index").plot.bar(
462     stacked=True, ax=ax, cmap=tab20_sido,
463 ).legend(
464     [
465         "Seoul",
466         "Gyeonggi",
467         "Incheon",
468         "Gangwon",
469         "Daejeon/Sejong/Chungcheong",
470         "Busan/Ulsan/Gyeongsangnam",
471         "Daegu/Gyeongsangbuk",
472         "Gwangju/Jeolla",
473         "Jeju",
474     ],
475     loc="center left",
476     bbox_to_anchor=(1.0, 0.5),
477 )
478
479 ax.set_xticks(range(n_clusters))
480 ax.set_xticklabels(range(1, n_clusters + 1))
481
482 ax.set_xlim(0, 1)
483 ax.yaxis.set_major_formatter(PercentFormatter(1))

```

```

484 fig.savefig("biclust_sido.png", bbox_inches="tight", pad_inches=0)
485 fig.savefig("biclust_sido.pdf", bbox_inches="tight", pad_inches=0)
486
487
488 # In[317]:
489
490
491 df_clust.Area
492
493
494 # In[318]:
495
496
497 pd.cut(df_clust.Area, [0, 50, 60, 85, np.inf])
498
499
500 # In[319]:
501
502
503 pd.crosstab(
504     df_clust.cluster, pd.cut(df_clust.Area, [0, 50, 60, 85, np.inf]), normalize="index"
505 )
506
507
508 # In[364]:
509
510
511 from matplotlib.ticker import PercentFormatter
512
513 fig = plt.figure(figsize=(5, 5), dpi=300)
514 ax = fig.gca()
515
516 pd.crosstab(
517     df_clust.cluster, pd.cut(df_clust.Area, [0, 50, 60, 85, np.inf]), normalize="index"
518 ).plot.bar(stacked=True, ax=ax).legend(loc="center left", bbox_to_anchor=(1.0, 0.5))
519
520 ax.set_xticks(range(n_clusters))
521 ax.set_xticklabels(range(1, n_clusters + 1))
522
523 ax.set_xlim(0, 1)
524 ax.yaxis.set_major_formatter(PercentFormatter(1))
525 fig.savefig("biclust_area.png", bbox_inches="tight", pad_inches=0)
526 fig.savefig("biclust_area.pdf", bbox_inches="tight", pad_inches=0)
527

```

528  
529 # # 면적  
530 #  
531 # 대다수는 국민주택 규모가 가장 많다.  
532 # 3, 6, 7, 15에서 소형 이하가 (< 60) 과반을 차지한다.  
533 # 10, 11, 13은 대형 (85 초과) 평면이 과반이다.  
534 #  
535 # 몇몇 예외 말고는 면적 규모에 따른 분류가 이루어지지 않았다.  
536 # 다양한 규모의 단위평면에서 나타나는 거시적인 변화를 나타낸다.  
537  
538 # # 총평  
539 #  
540 # 시기별로 잘 분류함  
541 #  
542 # 1990년대까지의 아파트 평면은 1--7번, 2000년대는 4--14번, 2010년대는 12--16번으로 중첩되어 나타남.  
543 # 하나의 평면 유형이 여러 시기에 걸쳐 나타났다가 사라지는 모습을 보여줌.  
544 #  
545 # 대체로 시기 구분과 관련없는 지역이나 면적 규모에 따른 차이는 잘 학습되지 않았음.  
546 #  
547 # 시기적으로 의미 있는 규모 변화는 잘 파악이 되었음.  
548 # 15번은 원룸형 도시형생활주택 (2009년 처음 등장)을 잘 찾아냄.

## 67\_VGG\_CAM\_visualization.py

```
1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # In[1]:
5
6
7 # import tensorflow as tf
8 import tensorflow.keras.backend as K
9
10 from tensorflow.keras.models import Sequential, Model
11
12 import cv2
13 import matplotlib.pyplot as plt
14 import numpy as np
15 import pandas as pd
16
17
18 # In[2]:
19
20
21 path_all_tfrecord = "fp56.tfrecord"
22
23
24 # In[3]:
25
26
27 dir_from = "/data/fp_img_processed/"
28
29
30 # In[4]:
31
32
33 dir_model = "vgg_cam/"
34 path_best = dir_model + "model-17-1.17-53.3%.hdf5"
35 path_best
36
37
38 # # model
39
40 # In[5]:
41
42
43 from fp_tensorflow import create_pair_56_dataset, create_single_dataset
```

```

44 from fp_tensorflow import create_vgg_5y_model
45
46 model = create_vgg_5y_model()
47 model.load_weights(path_best)
48 model.summary()
49
50
51 # # class activation map
52
53 # In[6]:
54
55
56 # Get the 512 input weights to the softmax.
57 class_weights = model.layers[-1].get_weights()[0]
58
59
60 # In[7]:
61
62
63 class_weights.shape
64
65
66 # In[8]:
67
68
69 class_weights.mean(), class_weights.std()
70
71
72 # In[9]:
73
74
75 def get_fp_output(fp, model=model):
76     final_conv_layer = model.get_layer("conv5_3")
77     get_output = K.function(
78         [model.layers[0].input], [final_conv_layer.output, model.layers[-1].output]
79     )
80
81     conv_output, prediction = get_output(np.expand_dims(fp, 0))
82     return np.squeeze(conv_output, axis=0), np.argmax(prediction)
83
84
85 # In[10]:
86
87

```

```

88     def get_fp_cam(fp, model=model):
89         class_weights = model.layers[-1].get_weights()[0]
90
91         conv_output, prediction = get_fp_output(fp, model)
92         true_class_weights = class_weights[:, prediction]
93
94         cam = np.zeros(dtype=np.float32, shape=conv_output.shape[0:2])
95         for i, w in enumerate(true_class_weights):
96             cam += w * conv_output[:, :, i]
97
98         return cam
99
100    # # biclust CAM
101
102    # In[11]:
103
104
105    biclusts = np.loadtxt("biclust_col.txt", int)
106    biclusts
107
108
109    # In[12]:
110
111
112    def get_biclust_cam(fp, biclust, model=model, labels=biclusts):
113        conv_output, _ = get_fp_output(fp, model)
114
115        return conv_output[..., biclusts == biclust].sum(axis=2)
116
117
118    # # plot
119
120    # In[13]:
121
122
123    def plot_bgr(img):
124        fig = plt.figure(figsize=(2, 2), dpi=300)
125        plt.axes().axis("off")
126        plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
127        plt.tight_layout()
128
129
130    # In[14]:
131
```

```

132
133 def plot_rgb(img):
134     fig = plt.figure(figsize=(2, 2), dpi=300)
135     plt.axes().axis("off")
136     plt.imshow(img)
137     plt.tight_layout()
138
139
140 # In[15]:
141
142
143 def plot_gray(img, cmap=plt.cm.gray):
144     fig = plt.figure(figsize=(2, 2), dpi=300)
145     plt.axes().axis("off")
146     plt.imshow(img, cmap=cmap)
147     plt.tight_layout()
148
149
150 # # run
151
152 # In[16]:
153
154
155 from floorplan_analysis import read_mono_from_image_unicode
156 from floorplan_analysis import fp_float_from_mono
157 from floorplan_analysis import pad_fp
158
159
160 # In[17]:
161
162
163 mono = read_mono_from_image_unicode(dir_from + "2888_118A" + ".png")
164 fp_full = fp_float_from_mono(mono)
165 fp = pad_fp(fp_full, 56, 56)
166
167 conv_output, prediction = get_fp_output(fp)
168
169
170 # In[18]:
171
172
173 fp_full.shape
174
175

```

```

176 # In[19]:
177
178 conv_output.shape, prediction.shape
180
181
182 # In[20]:
183
184
185 prediction
186
187
188 # In[21]:
189
190
191 cam = get_fp_cam(fp)
192 cam = cv2.resize(cam, (56, 56))
193 cam /= cam.max()
194 cam[cam <= 0] = 0
195
196 heatmap = cv2.applyColorMap(np.uint8(255 * cam), cv2.COLORMAP_VIRIDIS)
197 # heatmap[cam < 0.2] = 0
198 plot_bgr(heatmap)
199
200
201 # In[22]:
202
203
204 cam = get_biclust_cam(fp, 3)
205
206 cam = cv2.resize(cam, (56, 56))
207 print(cam.max())
208 cam /= cam.max()
209 cam[cam <= 0] = 0
210
211 heatmap = cv2.applyColorMap(np.uint8(255 * cam), cv2.COLORMAP_VIRIDIS)
212 # heatmap[cam < 0.4] = 0
213 plot_bgr(heatmap)
214
215
216 # In[23]:
217
218
219 def visualize_fp(fps):

```

```

220     # adjusted for different luminance
221     channel_to_rgba = np.array(
222         [
223             [0.0, 0.0, 0.0, 0.0], # wall to black L0
224             [0.0, 0.33, 0.0, 0.0], # entrance to green L30
225             [1.0, 0.25, 0.0, 0.0], # LDK to red L57
226             [0.83, 0.87, 0.0, 0.0], # bedroom to yellow L85
227             [0.0, 0.26, 1.0, 0.0], # balcony to blue L40
228             [0.0, 0.81, 0.76, 0.0], # bathroom to cyan L75
229         ]
230     )
231
232     # make colors subtractive
233     channel_to_rgba[:, 0:3] -= 1
234
235     # put it on white
236     fps_rgba = np.clip(
237         np.array([1.0, 1.0, 1.0, 1.0]) + (np.array(fps) @ channel_to_rgba), 0, 1
238     )
239     return fps_rgba.astype(np.float32)
240
241
242 # In[24]:
243
244
245 rgba = visualize_fp(fp_full)
246 plot_rgb(rgba)
247
248
249 # In[25]:
250
251
252 def visualize_fp_cam(fp):
253     fp_rgba = visualize_fp(fp)
254     fp_light = cv2.cvtColor(fp_rgba, cv2.COLOR_RGB2Lab)[:, :, 0] / 100
255
256     fp_pad = pad_fp(fp, 56, 56)
257
258     cam = get_fp_cam(fp_pad)
259     cam = cv2.resize(cam, (56, 56))
260     cam /= cam.max()
261     cam[cam <= 0] = 0
262
263     heatmap = cv2.applyColorMap(np.uint8(255 * cam), cv2.COLORMAP_VIRIDIS)

```

```

264     heatmap = pad_fp(heatmap, fp_light.shape[1], fp_light.shape[0])
265     heatmap[fp_light == 0] = 0
266     heatmap = heatmap.astype(np.float32) / 255
267
268     return 0.7 * heatmap + 0.3 * np.expand_dims(fp_light, 2)
269
270
271 # In[26]:
272
273
274 def visualize_biclust_cam(fp, biclust):
275     fp_rgba = visualize_fp(pad_fp(fp, max(56, fp.shape[1]), max(56, fp.shape[0])))
276     fp_light = cv2.cvtColor(fp_rgba, cv2.COLOR_RGB2Lab)[:, :, 0] / 100
277
278     fp_pad = pad_fp(fp, 56, 56)
279
280     cam = get_biclust_cam(fp_pad, biclust)
281     cam = cv2.resize(cam, (56, 56))
282     cam /= cam.max()
283     cam[cam <= 0] = 0
284
285     heatmap = cv2.applyColorMap(np.uint8(255 * cam), cv2.COLORMAP_VIRIDIS)
286     # heatmap = pad_fp(heatmap, fp_light.shape[1], fp_light.shape[0])
287     heatmap = pad_fp(heatmap, max(56, fp_light.shape[1]), max(56, fp_light.shape[0]))
288     heatmap = heatmap.astype(np.float32) / 255
289
290     return 0.7 * heatmap + 0.3 * np.expand_dims(fp_light, 2)
291
292
293 # In[27]:
294
295
296 plot_bgr(visualize_fp_cam(fp_full))
297
298
299 # In[28]:
300
301
302 plot_bgr(visualize_biclust_cam(fp_full, 3))
303
304
305 # # process representative floorplans
306
307 # In[29]:

```

```

308
309
310 df = pd.read_csv("biclust.csv")
311 df["area_group"] = pd.cut(df.Area, [0, 50, 60, 85, np.inf], labels=False)
312 df
313
314
315 # In[30]:
316
317
318 df_sample = df.groupby(["cluster", "area_group"]).sample(frac=0.005, random_state=1106)
319 df_sample = df_sample.sort_values(["cluster", "area_group", "year"])
320 df_sample
321
322
323 # In[31]:
324
325
326 pd.crosstab(df_sample.cluster, df_sample.area_group)
327
328
329 # In[32]:
330
331
332 pd.crosstab(df_sample.cluster, df_sample.area_group).max(axis=0)
333
334
335 # In[33]:
336
337
338 widths = np.asarray([3, 4, 8, 7])
339
340 coords_col = np.insert(np.cumsum(widths), 0, 0)[-1]
341 coords_col
342
343
344 # In[34]:
345
346
347 heights = np.maximum(
348     np.ceil(
349         pd.crosstab(df_sample.cluster, df_sample.area_group).to_numpy() / widths
350     ).astype(int),
351     1,

```

```
352     ).max(axis=1)
353     heights
354
355
356     # In[35]:
357
358
359     coords_row = np.insert(np.cumsum(heights), 0, 0)[-1]
360     coords_row
361
362
363     # In[36]:
364
365
366     sum(heights)
367
368
369     # In[37]:
370
371
372     sum(widths)
373
374
375     # 총 31줄, 19열
376
377     # In[38]:
378
379
380     u = 84 # unit size
381     flip = False
382
383
384     # In[39]:
385
386
387     if not flip:
388         img_size = (sum(heights) * u, sum(widths) * u)
389     else:
390         img_size = (sum(widths) * u, sum(heights) * u)
391
392
393     # In[40]:
394
395
```

```

396     img_size
397
398
399 # In[41]:
400
401
402 img = np.ones(img_size + (3,), np.float32)
403 # img = np.zeros(img_size + (3,), np.float32)
404
405
406 # In[42]:
407
408
409 plot_bgr(pad_fp(visualize_biclust_cam(fp_full, 3), u, u, 1))
410
411
412 # In[43]:
413
414
415 df_sample[(df_sample.cluster == 0) & (df_sample.area_group == 2)]
416
417
418 # In[44]:
419
420
421 df_sample[(df_sample.cluster == 0) & (df_sample.area_group == 2)].ID.iloc[1]
422
423
424 # In[45]:
425
426
427 for ir, rr in enumerate(coords_row):
428     for ic, cc in enumerate(coords_col):
429         df_clust = df_sample[(df_sample.cluster == ir) & (df_sample.area_group == ic)]
430         for i in range(len(df_clust)):
431             r = i // widths[ic]
432             c = i - r * widths[ic]
433             id_ = df_clust.iloc[i].ID
434             clust = df_clust.iloc[i].cluster
435
436             img[
437                 (rr + r) * u : (rr + r + 1) * u, (cc + c) * u : (cc + c + 1) * u
438             ] = pad_fp(
439                 visualize_biclust_cam(

```

```

440     fp_float_from_mono(
441         read_mono_from_image_unicode(dir_from + id_ + ".png")
442     ),
443     clust,
444     ),
445     u,
446     u,
447     1,
448 )
449
450
451 # In[46]:
452
453
454 fig = plt.figure(figsize=(11, 13), dpi=300)
455 ax = fig.gca()
456 im = plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
457 ax.set_xticks((coords_col + widths / 2) * u)
458 ax.set_xticklabels(
459     [
460         "One-room\nn($\\leq50\\mathrm{m^2}$)",
461         "Small\nn($\\leq60\\mathrm{m^2}$)",
462         "Medium\nn($\\leq85\\mathrm{m^2}$)",
463         "Large\nn($>85\\mathrm{m^2}$)",
464     ]
465 )
466 ax.set_yticks((coords_row + heights / 2 + 1 / 6) * u)
467 ax.set_yticklabels(range(1, biclusters.max() + 2))
468
469 ax.vlines(coords_col * u, 0, heights.sum() * u - 1, colors="k", lw=0.3)
470 ax.hlines(coords_row * u, 0, widths.sum() * u - 1, colors="k", lw=0.3)
471
472 fig.savefig("bam.png", bbox_inches="tight", pad_inches=0)
473 fig.savefig("bam.pdf", bbox_inches="tight", pad_inches=0)
474
475
476 # In[47]:
477
478
479 df_sample[(df_sample.cluster == 0)]
480
481
482 # 101160_113E
483 # 103915_112C

```

```

484 # 104127_107B
485 # 107903_113G
486 # 108838_117B
487
488 # In[48]:
489
490
491 def plot_bgr_scale(img):
492     size_x, size_y = img.shape[:2]
493     fig = plt.figure(figsize=(2 * size_x / 112, 2 * size_y / 112), dpi=300)
494     plt.axes().axis("off")
495     plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
496     plt.tight_layout()
497
498
499 # In[49]:
500
501
502 ir, ic, i = 15, 0, 0
503 u_single = 84 # 56 84 112
504 df_clust = df_sample[(df_sample.cluster == ir) & (df_sample.area_group == ic)]
505 id_ = df_clust.iloc[i].ID
506 print(id_)
507 clust = df_clust.iloc[i].cluster
508 plot_bgr_scale(
509     pad_fp(
510         visualize_biclust_cam(
511             fp_float_from_mono(read_mono_from_image_unicode(dir_from + id_ + ".png")),
512             clust,
513         ),
514             u_single,
515             u_single,
516             1,
517     )
518 )
519
520
521 # In[53]:
522
523
524 def plot_bams(id_, types):
525     print(id_)
526     fp = fp_float_from_mono(read_mono_from_image_unicode(dir_from + id_ + ".png"))
527     size_y, size_x = np.fmax(fp.shape[:2], 56)

```

```

528
529     clust_name = [
530         "8090-1",
531         "8090-2",
532         "8090-3",
533         "9000-1",
534         "9000-2",
535         "9000-3",
536         "9000-4",
537         "00-1",
538         "00-2",
539         "00-3",
540         "00-4",
541         "0010-1",
542         "0010-2",
543         "0010-3",
544         "10-1",
545         "10-2",
546     ]
547     clusts = [type - 1 for type in types]
548
549     fig, axs = plt.subplots(1, len(clusts), figsize=(11 / 4 * len(clusts), 5), dpi=300)
550
551     for i, clust in enumerate(clusts):
552         ax = axs[i]
553         img = pad_fp(visualize_biclust_cam(fp, clust), size_x, size_y, 1)
554         ax.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
555         ax.axis("off")
556
557         # title on bottom
558         # ax.set_title(clust_name[clust], y=-size_x / 56 / 10)
559
560         # title on top
561         ax.set_title(clust_name[clust], y=1)
562
563     plt.tight_layout()
564     return fig
565
566
567     # # 1980년대: 판상형, 복도형
568
569     # In[54]:
570
571

```

```

572 df_sample[
573     (df_sample.year >= 1980)
574     & (df_sample.year < 1990)
575     & (df_sample.cluster.isin([0, 1, 2]))
576     & (df_sample.area_group == 2)
577 ]
578
579
580 # 복도형 중형
581 #
582 # 서울 구로 주공1차, 73.08㎡, 1986년
583
584 # In[55]:
585
586
587 fig = plot_bams("137_96", [1, 2, 3])
588 # fig.savefig("bam.png", bbox_inches="tight", pad_inches=0)
589 # fig.savefig("bam.pdf", bbox_inches="tight", pad_inches=0)
590
591
592 # In[56]:
593
594
595 df_sample[
596     (df_sample.year >= 1980)
597     & (df_sample.year < 1990)
598     & (df_sample.cluster.isin([0, 1, 2]))
599     & (df_sample.area_group == 3)
600 ]
601
602
603 # 복도형 대형
604 #
605 # 서울 압구정 한양7차, 106.22㎡, 1981년
606
607 # In[57]:
608
609
610 fig = plot_bams("501_114A", [1, 2, 3])
611 # fig.savefig("bam.png", bbox_inches="tight", pad_inches=0)
612 # fig.savefig("bam.pdf", bbox_inches="tight", pad_inches=0)
613
614
615 # # 1990년대: 판상형, 계단식형

```

```

616
617 # In[58]:
618
619
620 df_sample[(df_sample.cluster.isin([3])) & (df_sample.area_group == 2)]
621
622
623 # 3LDK 중형
624 #
625 # 천안 일성3차, 84.82㎡, 1994년
626
627 # In[59]:
628
629
630 id_ = "7479_106"
631
632 fig1 = plot_bams(id_, [1, 2, 3])
633 fig2 = plot_bams(id_, range(4, 7 + 1))
634
635 # fig.savefig("bam.png", bbox_inches="tight", pad_inches=0)
636 # fig.savefig("bam.pdf", bbox_inches="tight", pad_inches=0)
637
638
639 # In[60]:
640
641
642 df_sample[
643     (df_sample.year >= 1990)
644     & (df_sample.year < 2000)
645     & (df_sample.cluster.isin(range(7 + 1)))
646     & (df_sample.Rooms == 4)
647 ]
648
649
650 # 4LDK 대형
651 #
652 # 인천 연수 하나2차, 99.42㎡, 1994년
653
654 # In[61]:
655
656
657 id_ = "2292_116"
658
659 fig1 = plot_bams(id_, [1, 2, 3])

```

```

660 fig2 = plot_bams(id_, range(4, 7 + 1))
661
662 # fig.savefig("bam.png", bbox_inches="tight", pad_inches=0)
663 # fig.savefig("bam.pdf", bbox_inches="tight", pad_inches=0)
664
665
666 # # 2000년대: 밸코니, 코어 후면 배치
667
668 # In[62]:
669
670
671 df_sample[
672     (df_sample.year >= 2000)
673     & (df_sample.year < 2010)
674     & (df_sample.cluster.isin([9]))
675     & (df_sample.area_group == 2)
676 ]
677
678
679 # 팬상형
680 # 중형
681 #
682 # 경기
683 # 동화옥시존5차,
684 # 84.58㎡,
685 # 2005년
686
687 # In[63]:
688
689
690 id_ = "17566_118"
691
692 fig1 = plot_bams(id_, range(8, 11 + 1))
693 fig2 = plot_bams(id_, range(12, 14 + 1))
694
695
696 # # 2010년대: 탑상형, 원룸형
697
698 # In[64]:
699
700
701 df_sample[(df_sample.cluster.isin([13])) & (df_sample.area_group == 2)]
702
703

```

```

704 # 탑상형 중앙부
705 # 중형
706 #
707 # 서울 서초포레스타5,
708 # 84.4m2,
709 # 2014년
710
711 # In[65]:
712
713
714 id_ = "107903_112B3"
715
716 fig1 = plot_bams(id_, range(12, 14 + 1))
717 fig2 = plot_bams(id_, range(15, 16 + 1))
718
719
720 # In[66]:
721
722
723 df_sample[(df_sample.cluster.isin([15])) & (df_sample.area_group == 2)]
724
725
726 # 탑상형 단부 중형
727 #
728 # 천안 백석더샵,
729 # 84.25m2,
730 # 2016년
731
732 # In[67]:
733
734
735 id_ = "108523_111C"
736
737 fig1 = plot_bams(id_, range(12, 14 + 1))
738 fig2 = plot_bams(id_, range(15, 16 + 1))
739
740
741 # In[68]:
742
743
744 df_sample[
745     (df_sample.cluster.isin([15]))
746     & (df_sample.year >= 2010)
747     & (df_sample.area_group == 2)

```

```

748     ]
749
750
751     # 혼합형
752     # (L자형 주동 계단실형 코어)
753     #
754     # 세종 가락4단지이지더원,
755     # 79.59㎡,
756     # 2014년
757
758     # In[69]:
759
760
761     id_ = "107076_106C"
762
763     fig1 = plot_bams(id_, range(12, 14 + 1))
764     fig2 = plot_bams(id_, range(15, 16 + 1))
765
766
767     # In[70]:
768
769
770     df[(df.cluster.isin([14]))].Area.mean()
771
772
773     # In[71]:
774
775
776     df[(df.cluster.isin([14]))].Area.median()
777
778
779     # In[72]:
780
781
782     df[(df.cluster.isin([14])) & (df.year >= 2010) & (df.Area >= 23) & (df.Area <= 29)]
783
784
785     # 원룸형 도시형생활주택
786     #
787     # 서울 역삼대명밸리온,
788     # 23.62㎡,
789     # 2012년
790
791     # In[73]:

```

```
792
793
794     id_ = "104259_36G"
795
796     fig = plot_bams(id_, [3, 6, 15])
797
```



## **Abstract**

# Deep Learning Based Spatial Analysis Method for Korean Apartment Unit Plans

AHN Euisoon  
Department of Architecture  
The Graduate School  
Seoul National University

This study aims to develop a methodology for analyzing the configuration of residential spaces in Korean apartment unit plans using deep learning. The goal for the methodology is to be capable of performing quantitative analysis on the total number of Korean apartments.

Since apartments became the mainstream housing type in Korea, the explosive increase of apartments made it hard to study. As a result, the study on apartment unit plans has been fragmented by various criteria. Spatial analysis, including spatial syntax, is a quantitative analysis methodology for the structure of an architectural space. It provided the basis for an objective and reproducible analysis of Korean apartments. However, even in spatial analysis methodology, manual work required for analyzing each case is the limiting factor. Because of that, There has been no spatial analysis study on the total cases of Korean apartments since the late 1990s. To overcome this limitation, this study attempted to develop a methodology that can inductively learn the researcher's inten-

tion from data and apply it to perform analysis on the whole of Korean apartments. To develop such an analysis methodology, deep learning, one of the machine learning methodologies, was used in this study. Deep learning has the advantage of being able to learn complicated concepts using a deep neural network model.

In Chapter 2, this study reviewed previous research that applied deep learning methodology to architectural spaces. The review confirmed that CNN (convolutional neural network) models could be applied successfully to architectural space by representing it as a two-dimensional matrix. Also, studies showed that the rules that the deep learning model learned from the data could be analyzed through LDA (latent space analysis) or deconvolutional neural networks. However, the deep learning model learned for other architectural spaces could not be directly applied to Korean apartments. Therefore, to apply deep learning to Korean apartments for analysis, the review suggested that it is necessary to construct a dataset for the total number of Korean apartments.

In Chapter 3, the thesis developed an analysis methodology on the configuration of residential spaces in Korean apartment unit plans using deep learning. First, reflecting the review on spatial analysis and deep learning, the spatial configuration model was developed in the form of a two-dimensional image. The image represents the space as a two-dimensional grid and describes the residential characteristics of each point in the depth dimension. Next, this study selected as the deep learning methodology for analyzing the spatial configuration. The CAM (class activation mapping) methodology visualizes the relationship between the configuration types from the deep learning model and the layout of the unit

plans. With the CAM method, it is possible to visualize the planning characteristics of each configuration type.

The limitation of CAM is that it can only be applied to the classification set in advance by the researchers. This study extended it into the BAM (bicluster activation map) methodology that visualizes the relationship between inductively classified types and the unit plans. BAM classifies the configuration types of residential space by pairing nodes of the latent layer inside the deep learning model with unit plans that activate it using biclustering analysis.

Then, this study designed a process to examine the analysis methodology developed in this study. Using a deep learning model that trained on the classification of Korean apartment unit plan dataset, the unit plans were classified based on the characteristics of the spatial configuration that differentiate the period. Finally, This study compared the result with findings from previous studies and examined the effectiveness of the analysis methodology.

In Chapter 4, this study constructed the Korean apartment unit plan dataset, which is necessary for analyzing the evolution of the Korean apartment unit plans and verifying the analysis methodology through the process. The unit plans and associated data were collected from publically accessible data. The configuration of residential spaces, such as entrance, LDK (integration of living room, kitchen, and dining room), bedroom, balcony, and restroom, was extracted from the plans. Through the process, a dataset for the total available data of Korean apartments was constructed.

Next, the study regularized the unit plans for consistent analysis. The main facing, the position of the entrance, and the scale of the plan were normalized. Also, the analysis area for comparison between unit plans of different sizes was set. Associated data such as completion year and regional jurisdiction were normalized to fit as input data of the deep learning model. In particular, reflecting the number of previous studies that set the time period a decade or a half, the completion year, from 1969 to 2019, was divided into ten periods of 5 years. Through this process, about 50 thousand unit plans and associated data were constructed into the dataset.

Chapter 5 analyzed the changes through time in the spatial configuration of the Korean apartment unit plans applying the methodology developed in this study. The VGG-GAP model, the model selected in Chapter 3, was modified to fit the dataset built in Chapter 4. Then the model was trained on the classification of the unit plans by time periods. Through this process, the spatial configuration analysis model predicted 90.51% of 50,252 plans within 5-year error.

Next, the learned representation of the spatial configuration by the periods was analyzed using the BAM methodology developed in Chapter 3. The biclustering analysis between about 50 thousand unit plans and 512 latent representation nodes derived 16 spatial configuration types. Each type co-clustered unit plans and latent nodes that are activated by those. However, the correlation of the activation was high between the types of similar periods, which means that the relationship can not be interpreted as mutually exclusive.

The unit plans of each spatial configuration type were differentiated by

the time period. However, there was no significant difference in most types by region and size. BAM visualization on the model for each type showed that the models of different types learned various characteristics in the unit plans of Korean apartments at different time periods.

In Chapter 6, this study reconstructed the evolutionary process of Korean apartments by applying the spatial configuration type derived in Chapter 5 to the unit plans from different periods and various building types. Then, the results were compared with previous studies to verify the analysis methodology developed in this study. Ultimately, the study investigated the possibility of utilizing an inductive analysis methodology based on deep learning for research on architectural space.

The subject of the analysis is the unit plans of various building types from the 1980s to the 2010s, including corridor-type and staircase-type plans from flat-type and mixed-type buildings, different plans from tower-type buildings, and one-room-type (studio) urban-type housing plans. The activation areas for each type of the period were analyzed by BAM visualization on the plans. The results from the analysis showed that the types and their counterpart partial model learned the configuration in the staircase-type unit plans from the 1990s. Also, BAM derived unique configurations of the corridor-type plans and various plans from tower-type buildings. However, it showed the limit of ignoring the overall configuration when there are clearly distinguishable differences in the plans of the configuration type.

The evolutionary process of Korean apartments derived the analysis mostly consistent with the findings of previous studies, including standardization of the staircase-type planning, domination of fewer plan

types in flat-type buildings, trickle-down of the configuration from large to small plans, introduction of tower-type building, maximization of balcony area, and standardization of the “balcony extension” (modification of the balcony as indoor space). The results confirmed the possibility of utilizing the analysis methodology developed in this study for the research in the evolution of Korean apartments.

The unit plan analysis methodology developed in this study does not require a manual process for individual cases. The significance of this study is that the methodology enables analysis of every unit plans of Korean apartments. Also, by demonstrating that the deep learning model learned only by unit plans and completion year can learn configuration types of various Korean apartments, the methodology showed that it could analyze abstract criteria that are not directly related to the spatial configuration. This study offers new possibilities by establishes the theoretical foundation for further research, including but not limited to, analyzing the spatial configuration of Korean apartments based on various criteria, expansion of analysis methodology for architectural spaces other than apartments, and developing generative design methodology of Korean apartment unit plans.

**Keywords:** Apartment, Unit plan, Spatial configuration, Machine learning, Deep learning, Typology

**Student Number:** 2011-30177