깊이에 따른 중간 단계 분류기 내부 학습 경향 분석 및 고찰

성수진○, 차정원

창원대학교
{20153057, jcha}@changwon.ac.kr

Analysis and Study of Internal Learning Trend of Deep Classifier according to Depth

Su-Jin Seong○, Jeong-Won Cha
Changwon National University

요 약

딥러닝 모델은 자동으로 자질을 추출하고 추상화 하기 위해 깊은 은닉층을 가지며, 이전 연구들은 이러한 은닉층을 깊게 쌓는 것이 성능 향상에 기여한다는 것을 증명해왔다. 하지만 데이터나 태스크에 따라 높은 성능을 내는 깊이가 다르고, 모델 깊이 설정에 대한 명확한 근거가 부족하다. 본 논문은 데이터 셋에 따라 적합한 깊이가 다르다고 가정하고, 이를 확인하기 위해 모델 내부에 분류기를 추가하여 모델 내부의 학습 경향을 확인하였다. 그 결과 태스크나 입력의 특성에 따라 필요로 하는 깊이에 차이가 있음을 발견하였고, 이를 근거로 가변적으로 깊이를 선택하여 모델의 출력을 조절하여 그 결과 성능이 향상됨을 확인하였다.

주제어: 중간 단계 분류기, 심층 분류기, 딥러닝, 문장 분류

1. 서론

딥러닝 모델은 입력으로부터 다양한 자질들을 자동으로 추출하고 이들을 추상화하기 위해 깊은 은닉층(Hidden layer)을 가지고 있다. 은닉층을 추가했을 때 gradient vanishing 혹은 degradation이 발생할 위험이 높아지지만 입력의 복잡한 특징을 이해하는 능력 또한 높아져 결과적으로 성능을 향상시킬 수 있다는 것이 확인되었다.

특히 대용량 데이터셋인 ImageNet을 이용한 경진대회인 ICVRC(ImageNet Scale Visual Recognition Challenge)에서 성능을 갱신해온 모델들[1,2,3,4]은 깊이가 깊어지는 경향을 보인다. ResNet[4]의 경우 skip connection을 도입하여 레이어를 152개까지 쌓는데 성공했고 오류율 3.57을 달성하였다.

이렇게 이미지 분류 분야에서 ResNet이 높은 성능을 보였고, [5]는 ResNet이 텍스트 분류 문제에서도 이점을 가진다는 것을 보이고자 하였다. 결과적으로 깊이가 9, 17, 29인 모델을 구축하고 29개의 레이어를 사용한 모델이 가장 높은 성능을 보였다.

[1,2,3,4,5]에서 모델의 깊이를 늘리는 것이 이미지 분류와 텍스트 분류 문제의 성능 향상에 도움을 줄 수 있음을 확인하였다. 하지만 ImageNet에서 ResNet이 사용한 깊이는 152이고 [5]에서 텍스트에 대해 사용된 깊이는 29로 차이를 보였다. 이로부터 우리는 데이터셋에 따라 적합한 깊이의 차이가 있을 것이라 가정하였다.

적합한 깊이를 찾기 위해서는 다양한 깊이의 모델을 구축한 후 실험을 통해 증명하는 방법을 사용할 경우 시간과 노력이 너무 많이 요구된다. 또한 구축한 모델 중 최고 성능을 달성한 모델을 찾는다 하더라도 이 때의 깊이는 임의로 설정되었기 때문에 최고 성능을 갱신할 수 있는 다른 깊이의 존재에 대한 의구심을 가질 수 있다.

본 연구에서는 모델 내부의 학습 경향을 분석하기 위해 중간 단계 분류기를 추가하는 방법을 제안한다. 중간 단계 분류기를 추가함으로써 학습 단계별 예측 결과물을 확인할 수 있으며 이를 분석한 결과를 모델 개선 방향 설정의 근거로 활용할 수 있다.

2. 관련 연구

[2]는 3x3 필터의 컨볼루션 레이어만을 사용하여 19개의 레이어를 쌓고, 추가로 fully-connected 레이어를 3개 사용한 모델 VGGNet을 구축하여 ILSVRC에서 오류율 7.3을 달성하였다. 간단한 구조이지만 3개의 fully-connected 레이어로 인해 파라미터 수가 1억 개가 넘어가는 단점이 있다.

[4]는 VGGNet과 동일하게 3x3 필터의 컨볼루션 레이어만 사용하고, 관측치와 예측의 차이인 나머지(residual)를 학습하도록 하였다. 그리고 입력을 레이어의 출력에 더하는 연산을 추가하였다. 또한 skip connection을 추가하여 레이어를 깊게 쌓을 수 있도록 하였다. 결과적으로 깊이 152의 모델로 오류율 3.57을 달성하여 최고 기록을 갱신하였다.

[5]는 이미지에서 제안된 ResNet을 텍스트를 이용한

분류 태스크에 적용하고자 하였고, skip connection을 이용해 모델을 개발하여 [6]에서 사용한 대규모 데이터 셋을 대상으로 실험을 진행하였다. 깊이가 9, 17, 29인 모델에 대해 실험하였고 29개의 레이어를 사용했을 때 가장 높은 성능을 보임을 확인하였다.

[7]에서는 DNN(Deep Neural Network)에서 발생할 수 있는 overthinking에 대한 연구를 진행하였다. 이 문제를 이해하기 위해 forward pass 과정 중 예측 값이 어떻게 도출되는지를 추출하여 분석하였다. 이 분석으로부터 대부분의 간단한 입력은 low-level 분류기에서 분류가 가능하며, 이렇게 low-level에서 정답에 도달한 입력이 추가적인 레이어 연산을 통해 연산적 낭비와 노이즈를 발생시키는 것이 확인되었다.

본 연구는 ResNet을 적용하여 모델의 깊이를 늘리고, 중간 단계 분류기를 추가하여 심층 분류 모델로 학습된 레이어들이 어떤 특성을 갖는지 분석하는 것을 목표로 한다.

3. 제안 방법

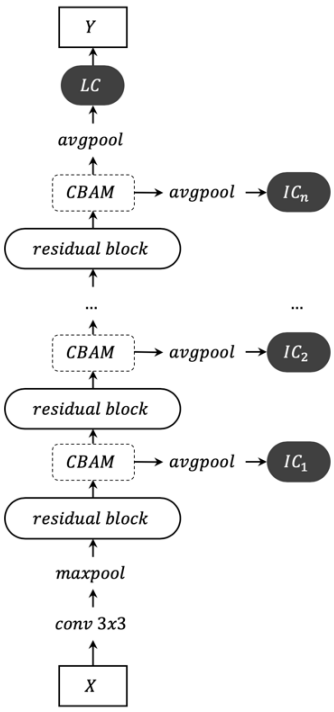

그림 1. 중간 단계 분류기가 추가된 ResNet인 Res-IC 구조도

그림 1은 ResNet의 중간 단계에 분류기(IC, Internal Classifier)를 추가한 구조를 보여준다.

Residual block은 2개의 컨볼루션 레이어로 구성되고, 컨볼루션 레이어의 필터는 모두 3x3으로 동일하다. 컨볼루션 레이어의 필터의 수를 2배로 증가시킬 때 레이어 당 복잡도를 유지하기 위해 stride를 2로 두어 down-sampling을 수행한다. 이러한 bolttleneck 부분에서 값이 줄어들기 전 중요한 자질의 가중치를 보정하기 위해 CBAM(Convolutional Block Attention Module)[8]을 추가한다.

중간 단계 분류기는 각 CBAM의 출력을 입력 받아 분류를 수행한다. 이 분류기들의 손실값은 최종 분류기(LC, Last Classifier)의 손실값과 함께 연산되어 모델의 파라미터를 갱신하는데 사용된다. 이 때 손실값이 중간 단계 분류기를 통해 해당 레이어에 직접적으로 전달되기 때문에 gradient vanishing 문제를 완화하고 파라미터 변화량이 커져 학습 속도가 줄어드는 이점을 가질 것으로 기대된다.

제안 모델은 결과적으로 최종 분류기의 성능을 향상시키는 것을 목표로 하기에, i 번째 중간 단계 분류기의 결과에 $\frac{i}{N}$의 가중치를 곱한 후 최종 분류기의 결과와 합하여 손실값으로 사용한다. 최종 손실값은 식 (1)과 같이 나타낼 수 있다.

$$L_{total} = L_{LC} + \sum_i^N L_{IC_i} \times \frac{i}{N} \quad (1)$$

$$L = -\frac{1}{N}\sum_j y_j \times \log(\hat{y_j}) \quad (2)$$

식 1의 L_{LC}는 최종 분류기의 손실값이고, L_{IC_i}는 i번째 중간 단계 분류기의 손실값이며, N 은 중간 단계 분류기의 개수이다. 손실값은 식(2)와 같이 cross-entropy로 계산하며, y_j 는 j 번째 입력에 대한 정답 카테고리, $\hat{y_j}$ 는 j 번째 입력에 대한 모델의 예측 카테고리이다.

4. 실험

4.1. 데이터

실험에 사용한 데이터는 범죄 발생 상황에 대한 내용으로 최대 132자로 제한된 동일한 입력 문장을 대상으로 한다. 태스크의 경우 입력이 속하는 사건 유형을 분류하는 것을 목표로 하는 Type과 입력이 포함하는 위험 정도를 예측하는 것을 목표로 하는 Risk로 나누어진다.

전체 데이터는 1,909,198개이며, 학습데이터로 1,403,421개, 평가 데이터로 389,840개, 검증 데이터로 115,937개를 사용한다. Type의 경우 20개의 카테고리를 가지며 가장 높은 빈도의 카테고리가 전체의 41.15%를 차지하고, 가장 적은 빈도의 카테고리는 0.10%를 차지한다. Risk의 경우 5개의 카테고리를 가지며 가장 높은 빈도의 카테고리가 전체의 40.02%를 차지하고, 가장 적은 빈도의 카테고리는 0.40%를 차지하여 두 태스크 모두 카테고리의 불균형이 심한 태스크이다.

4.2. 실험 모델

실험을 위해 8개의 Residual block과 CBAM을 이용해 모델을 구축하고, 각 CBAM의 출력에 average-pooling과, dropout과 softmax를 포함하는 fully-connected 레이어인 중간 단계 분류기 IC 를 추가한다. 모델의

구조도는 N이 8인 그림 1의 구조도와 같다. 7개의 중간 단계 분류기와 최종 분류기 LC는 입력의 차원만 각각 128, 128, 64, 64, 32, 32, 16, 16차원으로 다르고 동일한 구조로 동작한다. 이 모델을 ResNet[4]의 작명 방식을 따라 'Res-IC18'이라 명명한다.

4.3. 모델 성능 평가 지표

두 태스크는 입력 문장을 각각 20개, 5개의 카테고리로 구분하는 분류 문제이다. 따라서 학습된 모델의 성능을 Accuracy와 Macro average f1-score를 이용하여 측정한다. 식 (3)과 식 (4)의 K는 태스크의 전체 카테고리 개수를 의미한다.

$$Accuracy = \frac{\sum_i^K (TP_i + TN_i)}{\sum_i^K (TP_i + TN_i + FP_i + FN_i)} \quad (3)$$

$$F1 - Score = \frac{\sum_i^K \frac{2 \times Precision_i \times Recall_i}{Preicision_i + Recall_i}}{K} \quad (4)$$

5. 실험 결과 및 분석

5.1. Res-IC18의 중간 단계 분류기 결과 분석

표 1은 Res-IC18에 포함된 7개의 중간 단계 분류기와 최종 분류기의 성능을 나타낸다. Type의 경우 최종 분류기의 성능이 가장 높게 나타나며, Risk의 경우 6번째 중간 단계 분류기의 성능이 가장 높게 나타난다.

Overthinking 문제는 입력과 가까운 low-level 분류기에서 정답에 도달한 경우 추가로 수행되는 레이어 연산이 노이즈로 작용하여 오류가 발생하는 것을 말한다. 즉 중간 단계 분류기에서 가장 높은 성능을 보이고 그 후 분류기에서는 유사하거나 성능이 하락하는 경향을 보이는 것이다.

표 1 Res-IC18 중간 단계 분류기 별 성능

	Type		Risk	
	ACC	F1	ACC	F1
IC_1	0.87080	0.75695	0.82773	0.77595
IC_2	0.88317	0.80384	0.83734	0.79445
IC_3	0.89376	0.83300	0.84587	0.80649
IC_4	0.89486	0.83420	0.84700	0.80915
IC_5	0.89641	0.83676	0.84823	0.81142
IC_6	0.89653	0.83696	0.84837	0.81181
IC_7	0.89644	0.83787	**0.84869**	**0.81296**
LC	**0.89665**	**0.83804**	0.84862	0.81280

하지만 표 1의 경우 high-level로 진행될수록 성능이 향상되는 경향을 보인다. 이 결과로부터 우리는 모델이 적절한 깊이에 도달하지 못한 것으로 판단하고, residual block을 추가한 모델로 추가 실험을 진행하였다.

5.2. Res-IC18의 중간 단계 분류기 결과 분석

Res-IC18의 분석 결과를 근거로, residual block을 16개로 늘린 'Res-IC34' 모델을 구축하여 비교 실험을 진행하였다. Res-IC18과 모델 구성의 차이는 표 2에 작성하였다.

표 2 Res-IC18과 Res-IC34의 모델 구성 차이

	Res-IC18	Res-IC34
residual block	$\begin{pmatrix}3\times3,64\\3\times3,64\end{pmatrix}\times2$ $\begin{pmatrix}3\times3,128\\3\times3,128\end{pmatrix}\times2$ $\begin{pmatrix}3\times3,256\\3\times3,256\end{pmatrix}\times2$ $\begin{pmatrix}3\times3,512\\3\times3,512\end{pmatrix}\times2$	$\begin{pmatrix}3\times3,64\\3\times3,64\end{pmatrix}\times3$ $\begin{pmatrix}3\times3,128\\3\times3,128\end{pmatrix}\times4$ $\begin{pmatrix}3\times3,256\\3\times3,256\end{pmatrix}\times6$ $\begin{pmatrix}3\times3,512\\3\times3,512\end{pmatrix}\times3$
# of conv layer	16	32
# of CBAM	8	16
# of IC	7	15

표 3은 Res-IC18과 Res-IC34의 최종 분류기 성능을 나타내며, Block의 수를 늘린 Res-IC34의 성능이 더 향상되었음을 알 수 있다. Res-IC18과 차이를 확인하기 위해 Res-IC34의 중간 단계 분류기의 성능을 측정하여 표 4에 나타내었다.

표 3 Res-IC18과 Res-IC34의 성능 비교

	Type		Risk	
	ACC	F1	ACC	F1
Res-IC18	0.900	0.839	0.853	0.816
Res-IC34	0.902	0.842	0.855	0.816

표 4 Res-IC34의 중간 단계 분류기 별 성능

	Type		Risk	
	ACC	F1	ACC	F1
IC_1	0.82946	0.71507	0.80714	0.72560
IC_2	0.86343	0.74894	0.82987	0.77775
IC_3	0.8753	0.78628	0.83324	0.78408
IC_4	0.89141	0.82104	0.84523	0.80303
IC_5	0.89584	0.83559	0.84765	0.80925
IC_6	0.89665	0.83620	0.84908	0.81199
IC_7	0.89733	0.83748	0.84934	0.81224
IC_8	0.8986	0.84176	0.85021	0.81337
IC_9	0.8988	**0.84234**	0.85022	0.81347
IC_{10}	0.89887	0.84201	0.85031	0.81373
IC_{11}	**0.89889**	0.84193	0.85022	0.81361
IC_{12}	0.89881	0.84178	0.85026	0.81360
IC_{13}	0.89878	0.84171	0.85026	0.81360
IC_{14}	0.89883	0.84092	0.85041	0.81395
IC_{15}	0.89879	0.84082	0.85042	**0.81397**

LC	0.89869	0.84034	**0.85043**	0.81382

Risk의 경우 표 3의 결과와 같이 마지막 중간 단계 분류기의 성능이 가장 높은 것에 비해, Type의 성능은 Accuracy는 11번째 분류기, F1-Score는 9번째 분류기에서 최고점에 도달한다.

두 태스크 간의 학습 경향이 차이를 보이는 이유는 입력 문장이 각 태스크에 대해 갖는 복잡도가 다르기 때문이라 판단된다. Type은 20개의 카테고리를 갖고, Risk는 5개의 카테고리를 갖는다. 입력 문장이 자신이 포함된 카테고리에 대해 1개의 특징을 갖는다고 가정하면, Type 카테고리 중 하나에 포함되는 문장들의 특성은 Type에 대한 1개의 특성과 Risk에 대한 5개의 다른 특성을 포함할 것이다. 반면 Risk 카테고리 중 하나에 포함되는 문장들에서는 Risk에 대한 1개의 특성과 Type에 대한 20개의 다른 특성이 나타날 것이다. 즉 Risk 태스크가 Type 태스크보다 많은 특성을 분석해야하기 때문에 high-level 분류기에서의 분류 성능이 더 높게 나타난다.

Type에 대한 중간 단계 분류기의 성능을 보면 최고 성능을 달성한 이후 성능이 지속적으로 감소하는 것을 알 수 있다. 이는 최고 성능의 분류기 이후 연산이 노이즈로 작용하여 오류를 발생시킬 가능성이 있음을 보여준다. 즉 overthinking 문제가 발생하는 것이다.

Overthinking 문제를 해결하는 직관적인 방법은 모델의 깊이를 가장 성능이 높게 나오는 부분까지 조절하는 것이다. 불필요한 추가 연산을 제거함으로써 오류 발생 가능성과 연산 낭비를 줄일 수 있기 때문에 학습 속도도 감소할 것이고 성능 또한 향상될 것이라 기대된다.

하지만 overthinking으로 발생하는 오류를 줄이고자 전체 레이어 수를 조절하는 해당 방법은 high-level 레이어에서 올바르게 예측할 수 있는 복잡한 입력에 대한 처리 능력을 잃을 위험을 가진다.

이러한 가능성을 확인해보기 위해 입력 문장 길이의 차이가 10자씩 늘어날 때마다 복잡도가 증가한다고 가정하고, 오류 문장 길이에 대한 분석을 진행하였다.

그림 3과 그림 4는 Type에 대한 중간 단계 분류기의 오류 문장 길이 분포 차이를 나타낸 것으로 각각 30자 미만의 짧은 문장과 90자 이상의 긴 문장이 각 중간 단계 분류기의 전체 오류에서 차지하는 비율의 변화를 보여준다.

길이 별로 약간의 차이를 보이지만 대체로 짧은 문장의 경우 high-level 분류기로 갈수록 오류 비율이 높아지고, 긴 문장의 경우 반대의 경향을 보인다. 표 4의 결과에 근거하여 9번째나 11번째 block까지만 사용한다고 가정하면 90자 이상 110자 미만의 입력 문장에 대해 오류가 증가할 것이라 예상할 수 있다.

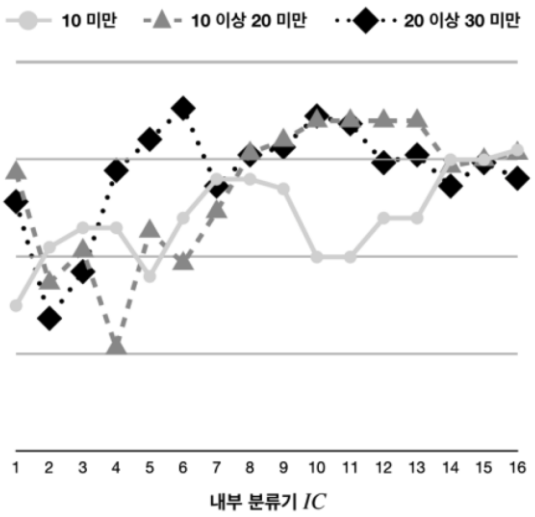

그림 2 Type에 대한 중간 단계 분류기의 오류 문장 중 짧은 문장의 비율

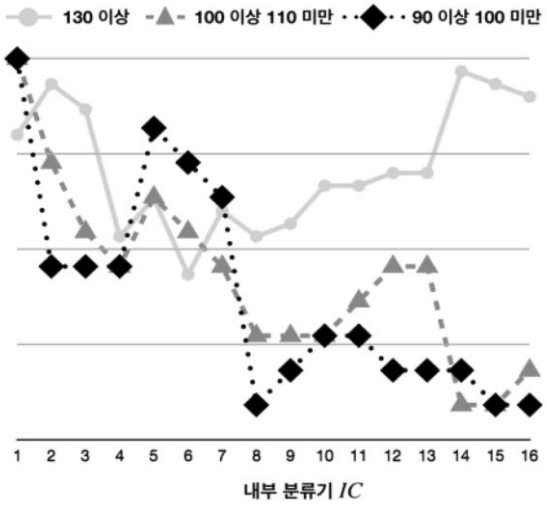

그림 3 Type에 대한 중간 단계 분류기의 오류 문장 중 긴 문장의 차지 비율

이러한 결과로 볼 때, 학습에 사용하는 residual block의 수를 입력 문장의 길이와 같은 입력의 복잡한 정도를 기준으로 가변으로 설정해준다면 더 나은 성능을 낼 수 있을 것이라 예상된다.

이에 우리는 입력의 복잡한 정도를 문장 길이로 정의하고, 입력 문장 길이 별로 R-IC34의 중간 단계 분류기의 성능을 확인하여 가장 높은 성능을 보이는 분류기의 결과를 최종 예측 결과로 출력하도록 하였다. 예를 들면 입력 문장이 10자 이상 20자 미만이면 2번째 중간 단계 분류기의 예측 카테고리를 모델의 예측 카테고리로 출력한다. 표 5는 Res-IC18, Res-IC34, 그리고 분류기를 가변적으로 선택하는 Res-IC34인 가변 Res-IC34의 성능을 나타낸다. 여기서 Max는 중간 단계 분류기의 성능 중 가장 높은 성능을 의미하고, ***LC***는 최종 분류기의 성능을 나타낸다.

표 5 가변 Res-IC34와의 성능 비교

		Type		Risk	
		ACC	F1	ACC	F1
Res-IC18	Max	0.89665	0.83904	0.84969	0.81296
	LC	0.89665	0.83804	0.84862	0.81280
Res-IC34	Max	**0.89889**	**0.84234**	0.85043	0.81397
	LC	0.89869	0.84034	0.85043	0.81382
가변 Res-IC34		0.89881	0.84127	**0.85458**	**0.81612**

표 5의 결과로, 가변으로 분류기를 선택하였을 때, Type의 가장 높은 중간 단계 분류기의 성능을 넘기지는 못했지만 Type과 Risk의 최종 분류기 성능보다 향상된 성능을 보임을 알 수 있었다. 이는 가변적으로 모델의 깊이를 선택하는 것이 모델의 성능 향상에 기여할 수 있다는 것을 보여준다.

6. 결론

본 논문은 residual block으로 깊이를 깊게 만든 모델에 중간 단계 분류기를 추가함으로써 블랙박스라고 표현되는 모델 내부의 학습 경향을 간접적으로 확인하고 분석하였으며, 분석 결과를 근거로 하여 모델 개선 방향 설정에 대한 타당성을 높였다.

주어진 태스크에 대한 모델을 중간 단계 분류기를 이용하여 분석한 결과, 데이터 혹은 태스크의 특성에 따라 레이어의 학습 경향이 다르게 나타나며 필요로 하는 깊이에 차이가 있음을 보여준다. 또한 최고 성능을 달성한 이후 연산이 노이즈로 작용하여 오류를 발생시키는 overthinking 문제가 발생함을 확인할 수 있었다.

문장의 길이를 복잡도의 기준으로 두고 오류 분포를 확인하였을 때 high-level 분류기일수록 짧은 문장의 오류 비율이 높았고, low-level 분류기에서는 긴 문장의 오류 비율이 높은 결과를 보였다. 이러한 결과로부터, overthinking 발생 위험을 줄이고자 레이어를 줄였을 때 high-level 레이어가 가진 복잡한 입력에 대한 처리 능력을 잃을 위험이 있다는 것을 알 수 있었다.

따라서 우리는 입력의 복잡한 정도에 따라 가변적으로 모델의 깊이를 선택하였을 때 성능이 향상될 것이라 가정하고 그 가능성을 실험을 통해 확인하였다.

본 연구에서 진행한 가변 깊이 모델 실험의 경우 입력의 복잡도를 간단하게 문장의 길이로 설정하였다. 짧은 문장이더라도 여러 특성을 가져 복잡할 수 있고, 긴 문장이더라도 각 문장이 유사한 특성을 가져 분석이 간단할 수 있기 때문에 이 또한 오류를 포함할 수 있다. 따라서 데이터의 특성을 파악하여 더 정확하게 입력의 복잡도를 정의한다면 성능 향상에 도움이 될 수 있을 것이다.

Acknowledgement

이 논문은 2019년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (No.2019-0-01755, 마취분야용 의료 딥러닝을 활용한 인공지능(ANES AI) 및 인터랙티브 OCS KIOSK 시스템 개발)

참고문헌

[1] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet Classification with Deep Convolutional Neural Networks, In NIPs, pp. 1097-1105, 2012

[2] K. Simonyan, A. Zisserman, Very Deep Convolutional Networks for Large-scale Image Recognition, In ICLR, 2015

[3] C. Szegedy, W. Liu, Y. jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going Deeper with Convolutions, In CVPR, pp. 770-778, 2015

[4] K. He, X.Zhang, S. Ren, J. Sun, Deep Residual Learnning for Image Recognition, In CVPR, 2018

[5] A. Conneau, H. Schwenk, L. Barrault, Y. Lecun, Very Deep Convolutional Network for Text Classification, In EACL, pp. 1107-1116, 2017

[6] K. He, X. Zhang, S. Ren, J. Sun, Delving Deep into Rectifiers: Surpassing Human-level Performance on imagenet classification. In ICCV, pp. 1026-1034, 2015

[7] Y. Kaya, H. Sanghyun, T. Dumitras, Shallow-Deep Networks: Understanding and Mitigating Network Overthinking, In ICML, 2019

[8] S. Woo, J. Park, J. Y. Lee, I. S. Kweon, Cbam: Convolutional block attention module, In ECCV, pp. 3-19, 2018