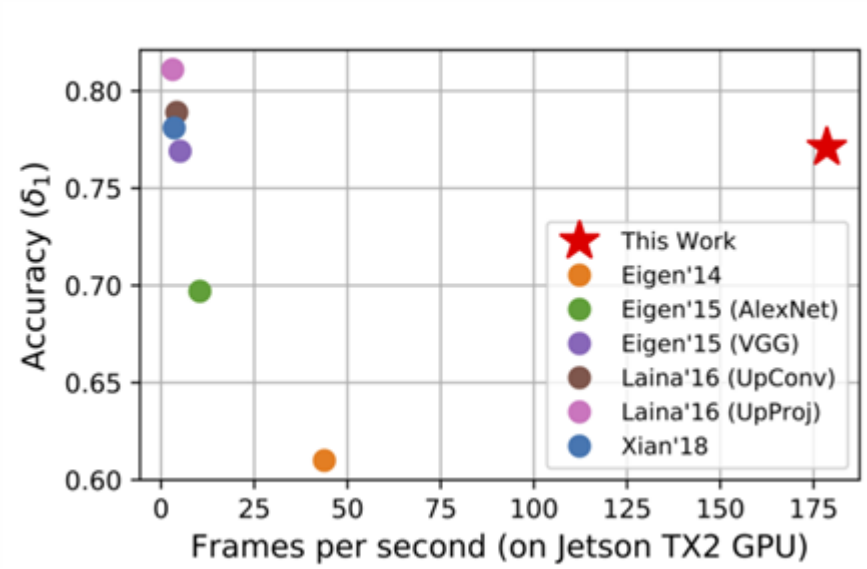
FastDepth: Fast Monocular Depth Estimation on Embedded Systems

로봇 작업에 있어 깊이 감지는 위치 결정, 지도 작성 및 장애물 감지와 같은 중요한 기능이다. 단일 RGB 이미지에서의 깊이 추정에 대한 관심이 크게 증가하고 있으며, 모노쿨러 카메라의 비교적 낮은 비용과 크기 때문에 더욱 그렇다. 그러나 최신 단일 뷰 깊이 추정 알고리즘은 상당히 복잡한 딥 뉴럴 네트워크를 기반으로 하며, 이는 임베디드 플랫폼에서 실시간 추론에는 느리다. 예를 들어 마이크로 항공 기기에 장착된 경우이다. 본 논문에서는 임베디드 시스템에서의 빠른 깊이 추정 문제에 대해 다룬다. 해당 논문은 효율적이고 가벼운 인코더-디코더 네트워크 구조를 제안하고, 계산 복잡성과 지연을 더 줄이기 위해 network pruning을 적용한다. 특히, 저지연 디코더의 설계에 중점을 둔다. 해당 논문의 방법론은 깊이 추정에서 이전 연구들과 유사한 정확도를 달성할 수 있으며 추론 속도는 한 차원 빠르다. 제안된 네트워크인 FastDepth는 NVIDIA Jetson TX2 GPU에서 178 fps 및 TX2 CPU만 사용할 때 27 fps로 실행되며, 활성 전력 소모가 10 W 미만이다. FastDepth는 NYU Depth v2 데이터 세트에서 거의 최신의 정확도를 달성합니다. 해당 논문은 마이크로 항공 기기에서 장착 가능한 임베디드 플랫폼에서 가장 낮은 지연과 최대 처리량을 가진 딥 뉴럴 네트워크를 사용한 실시간 단안 깊이 추정(monocular depth estimation)을 보여주고 있다.

깊이 감지는 매핑, 위치 결정 및 장애물 피하는 등 여러 로봇 작업에 필수적이다. 기존의 깊이 센서들은 일반적으로 거대하고 무거우며 전력 소모가 높다. 이러한 제한들은 깊이 센서를 소형 로봇 플랫폼(예: 마이크로 항공 및 소규모 지상 차량)에 적합하지 않게한다. 따라서 저렴한 비용, 소형 크기 및 높은 에너지 율성 때문에 monocular camera를 사용한 깊이 추정의 연구가 진행된다.

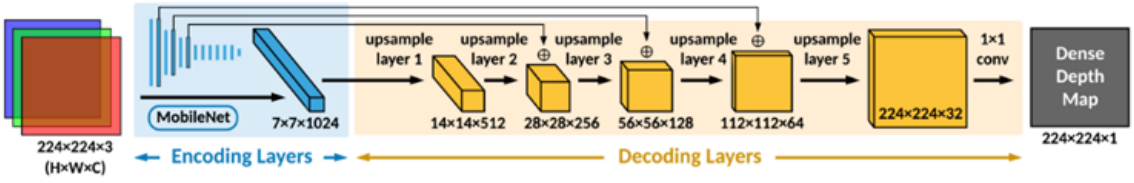
최신 깊이 추정(depth estimation) 알고리즘은 주로 딥러닝 기반 방법을 사용하며, 이러한 방법은 정확도에서 큰 개선을 이루지만 계산 복잡도가 증가한다. 빠르고 효율적이게 네트워크를 설계한 이전 연구는 주로 이미지 분류 및 객체 검출과 같은 작업을 위한 인코더 네트워크에 중점을 두었다. 이러한 응용 프로그램에서는 입력이 이미지(픽셀 기반)이고 출력이 레이블(객체 클래스 및 위치)로 감소된다. 출력이 밀집 이미지인 경우 깊이 추정과 같은 작업을 위한 인코더 및 디코더 네트워크의 효율적인 설계에는 별다른 노력이 들어가지 않는다. 특히, 각각의 디코딩 레이어에서 감소된 정보가 적게 발생하고 디코더의 출력이 고차원이기 때문에 디코더의 복잡성을 줄이는 추가적인 과제가 있다.



<그림 1 : NVIDIA Jetson TX2 GPU에서의 다양한 깊이 추정 알고리즘 정확도 vs 실행시간 >

이러한 문제를 다루기 위해 해당 논문은 임베디드 시스템에서 실행되는 저지연, 높은 처리량, 높은 정확도의 깊이 추정 알고리즘을 제안한다. 해당 논문은 저지연 설계에 중점을 둔 효율적인 인코더-디코더 네트워크 아키텍처를 제안한다. 해당 논문의 접근 방식은 인코더로 MobileNet을 사용하고 디코더에서는 깊이별 분리 컨볼루션을 사용하여 최근접 이웃 보간을 수행한다. 해당 논문은 최신 network pruning 기술인 NetAdapt를 적용하고 대상 임베디드 플랫폼에서 추론 런타임을 더 줄이기 위해 TVM 컴파일러 스택을 사용한다. 해당 논문은 낮은 지연 네트워크로 설계된 FastDepth가 NVIDIA Jetson TX2에서 실시간 깊이 추정을 수행할 수 있음을 보여준다. TX2 GPU에서 초당 120프레임 이상 작동하고 TX2 CPU에서 초당 25프레임 이상으로 작동하며, 유효 전력 소모량이 10W 미만이다. 도달한 처리량은 깊이 추정에 대한 이전 연구보다 한 차원 높은데, 정확도의 손실은 거의 없다. FastDepth는 NYU Depth v2에서 77.1%의 정확도를 달성한다.
해당 논문의 네트워크에서 얻은 낮은 지연과 높은 처리량은 현실적인 로봇 시스템에서 영감을 받았다. 여기에는 localization, mapping, motion planning, control 및 잠재적인 다른 perception tasks와 같은 다중 프로그램이 병렬로 실행되는 task가 포함된다. 각각의 프로그램은 일정량의 계산 자원을 필요로 한다. 따라서 CPU/GPU는 깊이 추정 임무에만 사용하고 있지 않다. 낮은 지연 네트워크 설계는 한정된 컴퓨팅 예산에서도 실시간 성능을 가능하게 하며 각 프로그램이 필요로 하는 계산 자원을 효율적으로 할당할 수 있다.
요약하자면, 해당 논문은 마이크로형 무인기에 탑재할 수 있는 임베디드 플랫폼에서 실시간 단안 깊이 추정을 달성하는 딥 뉴럴 네트워크를 소개한다. 이 네트워크는 최저 지연과 최대 처리량을 달성하며, 로봇 시스템에서 여러 작업이 동시에 실행될 때도 효율적으로 작동한다. 해당 논문에서 제안한 완전 컨볼루션 인코더-디코더 아키텍처는 그림 2에서 확인할 수 있다.

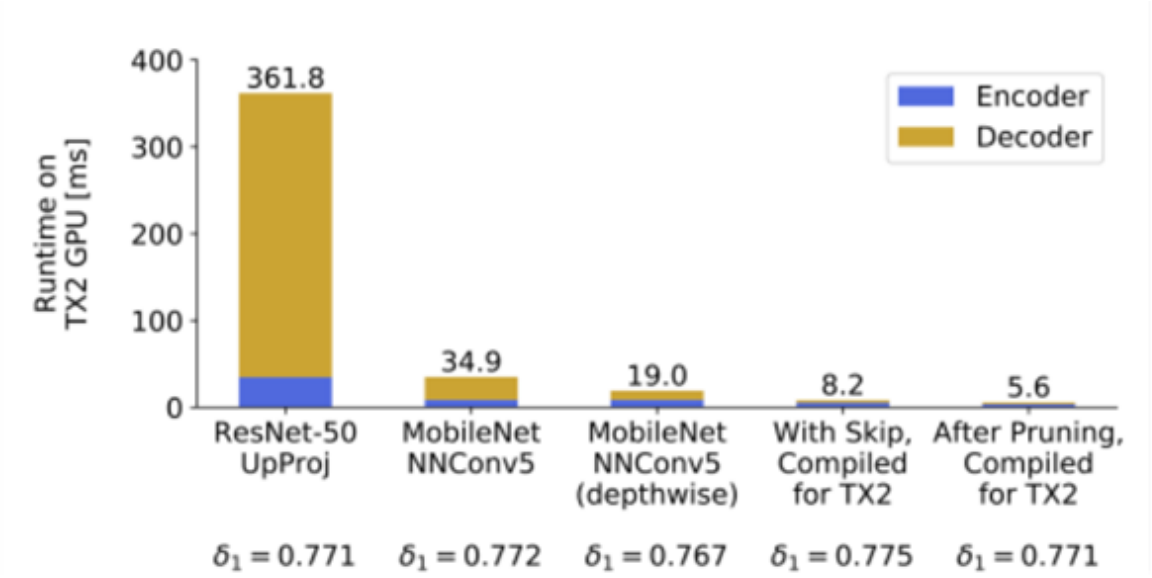
인코더는 입력 이미지에서 고수준의 저해상도 특징을 추출한다. 그런 다음 이러한 특징들은 디코더로 전달되어 점진적으로 업샘플링되고 정제되며 병합되어 최종 고해상도 출력 깊이 맵을 형성한다. 실시간으로 실행될 수 있는 깊이 추정 네트워크를 개발함에 있어서, 해당 논문은 인코더와 디코더 둘다 저지연 설계를 추구한다.
인코더 네트워크는 깊이 추정 네트워크에서 사용되는 인코더는 일반적으로 이미지 분류를 위해 설계된 네트워크이다. 널리 사용되는 선택지로는 VGG-16 및 ResNet-50가 있다. 그 모델들은 강한 표현력과 높은 정확도 때문에 인기가 있다. 그러나 이러한 네트워크 역시 높은 복잡성과 지연을 일으켜서 임베디드 시스템에서 실시간으로 실행되는 응용에는 적합하지 않다.



<그림 2 : 해당 논문에서 제안한 네트워크 구성>

해당 논문은 낮은 지연을 목표로 하여 인코더로 MobileNet이라는 효율적인 최신 네트워크를 선택했다. MobileNet은 depthwise decomposition를 사용하며, 이는 m×m×n 표준 컨볼루션 레이어를 m×m×n 깊이별 레이어와 1×1 포인트별 레이어로 분해한다. 깊이별 레이어의 각 필터는 단일 입력 채널과만 컨볼루션 연산을 해서 깊이별 레이어의 복잡성은 각 필터가 모든 입력 채널과 컨볼루션 연산하는 표준 컨볼루션 레이어보다 훨씬 낮다. 또한, 각 포인트별 필터는 단순히 1×1 커널이므로 포인트별 레이어가 수행하는 MACs의 수는 원래의 표준 컨볼루션 연산보다 m2 배 작다. 따라서 깊이 분해는 컨볼루션 연산의 복잡성을 크게 줄여주어 MobileNet이 ResNet 및 VGG와 같은 표준 컨볼루션 연산을 사용하는 네트워크보다 더 효율적이게 된다. 이는 지연 감소로 이어지게 된다.
디코더 네트워크는 디코더의 목적은 인코더의 출력을 병합하고 업샘플링하여 밀집 예측을 형성하는 것이다. 디코더의 중요한 설계 측면 중 하나는 사용되는 업샘플링 작업이다(예: 언풀링, 전치 컨볼루션 연산, 컨볼루션과 결합된 보간). 해당 논문의 디코더 네트워크 (NNConv5)는 5개의 연속된 업샘플 레이어와 마지막에 단일 pointwise layer로 구성되어 있다. 각 업샘플 레이어는 5 × 5 컨볼루션을 수행하고 출력 채널 수를 입력 채널 수에 대해 1/2로 감소시킵니다. 컨볼루션 후에는 중간 특징 맵의 공간 해상도를 두 배로 늘리는 최근접 이웃 보간이 이어진다. 컨볼루션 레이어에 의해 처리된 특징 맵의 해상도를 낮추기 위해 보간을 컨볼루션 이후에 수행한다. 해당 기술은 모든 컨볼루션 레이어의 복잡성을 더 낮추기 위해 깊이 분해를 사용하여 가볍고 빠른 디코더를 만들었다.
스킵 연결(Skip Connections)은 인코더 네트워크에는 일반적으로 공간 해상도를 점진적으로 줄이고 입력에서 더 높은 수준의 특징을 추출하기 위한 많은 레이어가 포함되어 있다. 인

코더에서 디코더로의 출력은 많은 이미지 세부 사항이 손실될 수 있는 저해상도 특징 집합이 되므로 픽셀별 데이터를 디코더에서 복원하기가 더 어려워진다. 스킵 연결을 사용하면 인코더의 고해상도 특징 맵에서 디코더 내의 특징으로 병합할 수 있다. 이것은 디코딩 레이어가 더 자세한 밀집 출력을 재구성하는 데 도움이 된다. 해당 기술은 MobileNet 인코더에서 디코더의 중간 세 개의 레이어로 스킵 연결을 통합했다. 특징 맵은 더 많은 특징 맵 채널의 수를 증가시키지 않기 위해 연결이 아닌 덧셈을 통해 결합된다.
해당 논문에서 제안한 네트워크 아키텍처는 완전 컨볼루션으로 이루어져 있으며, 인코더와 디코더에서 깊이 분해를 사용한다. 현재 깊이 분해 가능한 컨볼루션 레이어는 일반적으로 널리 사용되는 딥러닝 프레임워크에서 빠른 런타임을 위해 완전히 최적화되어 있지 않다. 이로 인해 깊이 분해 레이어로 얻을 수 있는 복잡성 감소를 하드웨어에서 런타임 감소로 변환하기 위해 하드웨어별 컴파일이 필요하다. 해당 논문에서는 Jetson TX2와 같은 임베디드 플랫폼에 배포하기 위해 제안된 네트워크 디자인을 컴파일하는 데 TVM 컴파일러 스택을 사용한다.
네트워크 지연을 더욱 줄이기 위해 해당 논문은 최신 기술인 NetAdapt 알고리즘을 사용하여 훈련 후에 network pruning을 수행한다. 훈련된 네트워크를 기반으로 시작하여 NetAdapt는 계산 복잡성을 줄이기 위해 특징 맵에서 중복 채널을 자동적으로 식별하고 반복적으로 제거한다. 각 반복에서 NetAdapt는 참조 네트워크에서 단순화된 일련의 네트워크 제안을 생성한다. 최적의 정확도-복잡성 균형을 갖는 네트워크 제안이 선택되어 다음 반복에서 참조 네트워크로 사용된다. 이 과정은 목표 정확도 또는 복잡성이 달성될 때까지 계속된다. 네트워크 복잡성은 간접 측정 항목 (예: MACs) 또는 직접적인 측정 항목 (예: 대상 하드웨어 플랫폼에서의 지연)으로 측정될 수 있다.
해당 논문은 네트워크를 훈련하고 그 정확도를 NYU Depth v2 데이터셋에서 공식 훈련/테스트 데이터 분할로 평가한다. 네트워크 훈련은 Sparse-to-dense와 유사하게 PyTorch에서 32비트 부동 소수점 정밀도로 구현되었다. 훈련에는 배치 크기 8과 학습률 0.01이 사용된다. 옵티마이저는 모멘텀이 0.9이고 가중치 감쇠가 0.0001인 SGD이다. 인코더 가중치(예: MobileNet 및 ResNet)는 ImageNet에서 사전 훈련되었다. 정확도는 상대 오차가 25% 이내인 예측된 픽셀의 백분율인 δ1과 RMSE(평균 제곱근 오차)로 측정된다. 평가에는 배치 크기 1이 사용되며 정밀도는 32비트 부동 소수점으로 유지된다. 주요 대상 플랫폼은 Jetson TX2의 max-N 모드이다.
해당 논문에서 제안된 방법론으로 얻은 결과는 그림 3에 요약되어 있다. ResNet-50 with UpProj는 기준선으로 사용되며, 이 네트워크는 Deeper depth prediction with 완전 컨볼루션 잔차 네트워크에서 설명된 아키텍처를 따른다. 이 기준선 네트워크의 실행 시간은 주로 디코더에 의해 결정된다. 해당 논문의 접근 방식에서 가장 즉각적이고 중요한 실행 시간 감소는 더 작고 계산적으로 간단한 디코더를 사용함으로써 나타난다. 그러나 MobileNet 인코더와 결합할 때에도 디코더가 여전히 네트워크 실행 시간을 지배한다. 디코더 내의 컨볼루션을 깊이 분해로 간소화함으로써 디코더의 실행 시간이 MobileNet과 더 가까워지기 시작한다. Jetson TX2 플랫폼을 대상으로 network pruning하고 컴파일하면 인코더와 디코더의 실행 시간이 더욱 줄어들어, 궁극적으로 기준선 대비 65배 정도의 총 추론 실행 시간 감소를 이루어내며 최대 178 fps의 매우 높은 처리량을 가능하게 한다.



<그림 3: 해당 논문의 접근 방식으로 얻은 추론 실행 시간>

제안된 모델의 정확도와 지연 시간 지표는 이전 연구와 비교하여 표 1에 요약되어 있다. 해당 논문은 CRFs와 같은 추가 처리를 포함하지 않은 딥러닝을 사용하는 깊이 추정 방법과 비교하여 평가다. 왜냐하면 추가 처리는 계산 및 실행 시간 비용을 발생시킨다.

<표 1: 이전 연구와 비교>

on NYU Depth v2	Input Size	MACs [G]	RMSE	δ_1	CPU [ms]	GPU [ms]
Eigen *et al.* [11]	228×304	2.06	0.907	0.611	307	23
Eigen *et al.* [16] (AlexNet)	228×304	8.39	0.753	0.697	1391	96
Eigen *et al.* [16] (VGG)	228×304	23.4	0.641	0.769	2797	195
Laina *et al.* [12] (UpConv)	228×304	22.9	0.604	0.789	2384	237
Laina *et al.* [12] (UpProj)	228×304	42.7	**0.573**	**0.811**	3928	319
Xian *et al.* [37]	384×384	61.8	0.660	0.781	4429	283
This Work	224×224	**0.37**	0.604	0.771	**37**	**5.6**

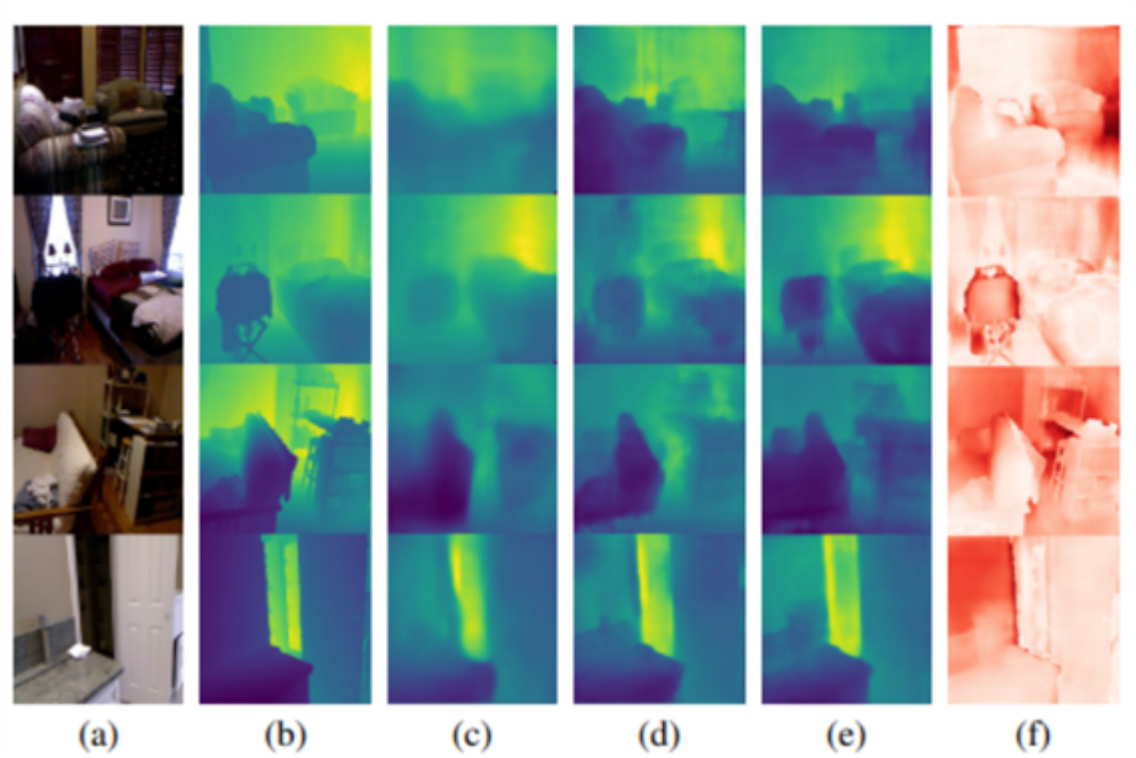
해당 논문은 모델을 TX2에서 max-N 모드로 실행할 때의 활성 전력 소비를 10 W 미만으로 측정한다. 또한 더 에너지 효율적인 max-Q 모드에서 TX2의 런타임 및 전력 소비 데이터를 보고한다. 표 2는 이러한 데이터를 요약한다. max-Q 모드에서 TX2를 사용할 경우, 해당 논

문의 모델은 CPU에서 거의 실시간 속도를 달성할 수 있으며 유효 전력 소비량이 5W 미만으로 유지되면서 GPU에서는 여전히 쉽게 실시간 속도를 초과한다.

<표 2: 해당 논문의 모델을 Jetson TX2에 배포할 때의 추론 실행 시간과 최대 전력 소비>

Platform	Runtime	Max Frame Rate	Power Consumption
TX2 GPU (max-N)	5.6 ms	178 fps	12.2 W (3.4 W idle)
TX2 GPU (max-Q)	8.2 ms	120 fps	6.5 W (1.9 W idle)
TX2 CPU (max-N)	37 ms	27 fps	10.5 W (3.4 W idle)
TX2 CPU (max-Q)	64 ms	15 fps	3.8 W (1.9 W idle)

그림 4는 NYU Depth v2 데이터셋의 이미지에 대한 저희 모델의 생성 결과를 시각화한다. 인코딩과 디코딩 레이어 간의 스킵 연결은 깊이 맵 출력의 선명도와 시각적 명확성을 향상시킨다. network pruning은 명확성을 보존하고 향상시키는 효과를 나타낸다. 또한 모델 출력과 실제 값 간의 차이를 시각화한 오류 맵을 보여준다. 오류는 경계 및 거리가 먼 객체에서 가장 높음을 주목해야 한다.
기존의 높은 정확도 접근 방식에서 일반적으로 사용되는 공통 인코더는 ResNet-50 이다. 해당 논문은 낮은 인코더 지연을 목표로 하여 ResNet-18과 MobileNet을 ResNet-50의 대안으로 생각한다. MobileNet과 ResNet 아키텍처에서는 마지막 평균 풀링 레이어와 완전 연결 레이어를 제거한다. 인코더를 고정된 디코더 구조와 호환되게 만들기 위해 ResNet 인코더 끝에 1×1 컨볼루션 레이어를 추가하여 모든 인코더 변형의 출력이 1024 채널과 7×7의 일관된 형태를 가지게 한다.
해당 논문은 세 가지 인코더 옵션을 서로 비교하고 표 3에서 보여준다. 보고된 실행 시간은 PyTorch에서 인코더 네트워크를 컴파일하고 실행하여 얻었다. ResNet-50 및 ResNet-18의 실행 시간은 TX2 GPU에서 실시간 속도(즉, 30 fps 이상)를 달성하는 데 너무 높아서 이러한 인코더가 유사한 지연을 가진 디코더와 결합되면 실시간 속도를 달성할 수 없다.



<그림 4: NYU Depth v2 데이터셋에서 깊이 추정 결과 시각화>

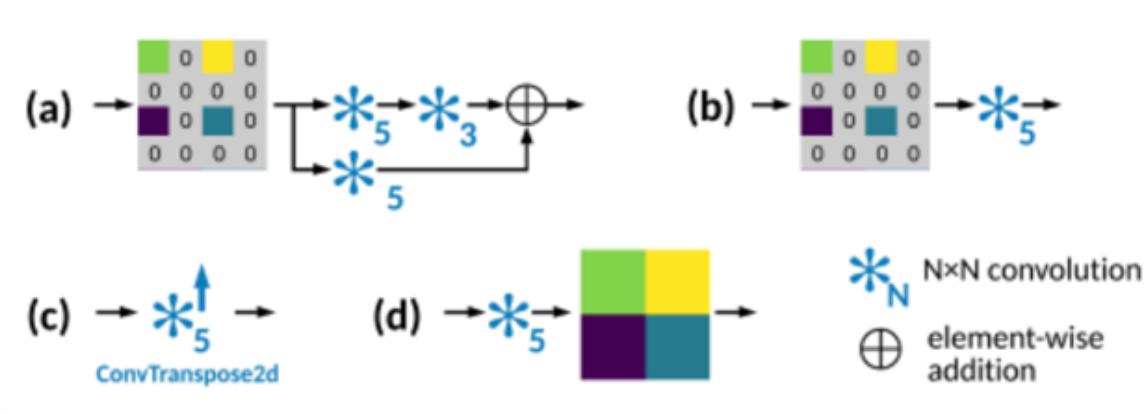
반면에 MobileNet은 정확도와 지연 사이에서 효율적으로 교환하며 GPU 실행 시간이 뚜렷하게 낮다. 따라서 해당 논문은 인코더로 MobileNet을 선택했다. 낮은 복잡성에도 불구하고 MobileNet은 TX2 CPU에서 ResNet-18에 비해 한 차원 느리다. 이는 아직 최적화되지 않은 딥러닝 프레임워크에서 깊이별 레이어의 구현에 기인하며, 이에 대한 대안 딥러닝 컴파일러의 필요성을 생각해볼 필요가 있다.

<표 3: 인코더 비교>

Encoder	Weights [M]	MACs [G]	RMSE [meters]	δ_1	CPU [ms]	GPU [ms]
ResNet-50	25.6	4.19	**0.568**	**0.800**	610	35.0
ResNet-18	11.7	1.84	**0.568**	0.782	**220**	15.2
MobileNet	**3.19**	**0.57**	0.579	0.772	3700	**8.7**

해당 논문은 여러 디코더 설계 측면(업샘플링, 깊이별 분해, 스킵 연결)을 고려한다.
Upsample Operation: 디코더에서 업샘플링하는 네 가지 방법이 있다. 각각의 특징은 아래

에 나열되어 있으며, 시각적 표현은 그림 5에 나와 있다.
(a) UpProj은 2×2 언풀링(제로 삽입) 후에 두 가지 분기된 잔차 구조가 이어지며, 이는 총 세 개의 컨볼루션을 계산한다(두 개의 5 × 5와 하나의 3 × 3).
(b) UpConv은 2×2 언풀링(제로 삽입) 후에 단일 5×5 컨볼루션이 이어진다.
(c) DeConv5는 5 × 5 커널을 사용한 전치 컨볼루션이다.
(d) NNConv5는 5 × 5 컨볼루션 후에 2 배 축소율로 가장 가까운 이웃 보간(nearest-neighbor interpolation)이 이어진다.



<그림 5: 다양한 업샘플링 작업의 시각적 표현 (a) UpProj, (b) UpConv, (c) DeConv5, (d) NNConv5>

해당 논문은 이러한 업샘플링 작업을 사용하여 구조를 5개의 디코딩 레이어로 고정한 네 가지 디코더 변형을 구현한다. 표 4에서 네 가지 디코더를 비교한다. UpProj는 각 업샘플 레이어 당 더 많은 컨볼루션을 갖기 때문에 가장 복잡하다. 가장 높은 δ1 정확도를 달성하지만 가장 느리다. UpConv는 UpProj보다 덜 복잡하며 더 빠르지만, 여전히 실시간 처리에는 CPU 및 GPU 실행 시간이 너무 느리다. DeConv5는 UpConv와 동일한 수의 가중치와 MAC를 갖고 있으며 CPU 및 GPU에서 뚜렷하게 빠르다. 그러나 출력에서 checkboard의 훼손을 도입할 수 있으며 이로 인해 정확도가 낮아진다. NNConv5는 UpConv 및 DeConv5보다 더 높은 δ1 정확도와 낮은 RMSE를 달성하며, 약간 낮은 GPU 실행 시간을 갖고 있다. 따라서 해당 논문은 NNConv5를 제안된 디코더로 선택한다.

<표 4: 디코더 비교>

Decoder	Weights [M]	MACs [G]	RMSE [meters]	δ_1	CPU [ms]	GPU [ms]
(a) UpProj [12]	38.1	28.0	0.599	**0.774**	3300	325
(b) UpConv [12]	**17.5**	12.9	0.591	0.771	1600	238
(c) DeConv5	**17.5**	12.9	0.596	0.766	**290**	31.0
(d) NNConv5	**17.5**	**3.21**	**0.579**	0.772	410	**26.2**

깊이별 분리 가능한 컨볼루션은 MobileNet을 인코더로 선택하고 NNConv5를 디코더로 선택한 후, 네트워크의 실행 시간이 디코더의 영향을 받는다(인코더-디코더 분해를 위해 그림 3 참조). 이로 인해 해당 논문은 디코더를 더욱 단순화하기로 결정했다. MobileNet에서 복잡성과 지연을 줄이는 깊이 분해와 유사하게, 디코더 내의 모든 컨볼루션을 깊이 분리 컨볼루션으로 대체한다. 표 5에서 볼 수 있듯이 깊이 분해된 디코더는 GPU에서 추론 실행 시간을 거의 절반으로 낮춘다. 그러나 MobileNet의 경우와 마찬가지로 디코더 내의 깊이별 레이어는 훈련 가능한 매개변수와 계산의 감소로 인해 약간의 정확도 손실을 일으킨다. 잃어버린 정확도의 일부를 복원하기 위해 인코딩 및 디코딩 레이어 간에 스킵 연결를 통합한다.

<표 5: 깊이 분해 디코딩 레이어와 스킵 연결이 네트워크 복잡성과 TX2 실행 시간에 미치는 영향>

MobileNet-NNConv5	Weights [M]	MACs [G]	RMSE [meters]	δ_1	CPU [ms]	GPU [ms]
with standard decoder	20.6	3.78	**0.579**	0.772	**4100**	34.9
with depthwise decoder	**3.93**	**0.74**	0.584	0.767	5200	**18.6**
depthwise & skip-concat	3.99	0.85	0.601	**0.776**	5500	26.8
depthwise & skip-add	**3.93**	**0.74**	0.599	0.775	5100	19.1

스킵 연결은 해당 논문에서 가산 및 연결형 스킵 연결를 둘다 고려한다. 연결형 스킵 연결은 디코더의 계산 복잡성을 증가시킨다. 왜냐하면 디코딩 레이어가 더 많은 채널을 갖는 피쳐 맵을 처리해야 하기 때문이다. 표 5에서는 이로 인해 δ1 정확도가 향상되었고 CPU 및 GPU 실행 시간이 뚜렷하게 증가한 것을 보여준다. 반면, 가산 스킵 연결를 사용하면 디코더의 채널 수가 변경되지 않아 추론 실행 시간에는 미미한 영향만 미치면서 거의 동일한 정확도 향상을 달성한다. 따라서 해당 논문은 최종 네트워크 설계에서 가산 스킵 연결를 사용한다. 그림 4(d)에서 보듯이 스킵 연결는 제안된 네트워크 설계가 출력하는 깊이 맵의 선명도와 시각적

명확성을 뚜렷하게 향상시킨다.
현재 딥러닝 프레임워크는 하드웨어별 최적화 수준이 다를 수 있는 프레임워크별 연산 라이브러리에 의존하고 있다. 해당 논문에서 제안한 네트워크 아키텍처는 인코더와 디코더 전체에 걸쳐 깊이별 레이어를 포함하고 있다. 현재 이러한 레이어는 흔히 사용되는 딥러닝 프레임워크에서 아직 완전히 최적화되지 않았다. 결과적으로 깊이 분해는 네트워크의 MAC 수를 크게 줄이지만 지연에는 유사한 감소가 반영되지 않는다. 표 6의 왼쪽 부분은 정확히 이를 강조한다. MobileNet-NNConv5의 TX2 CPU 실행 시간은 이미 MobileNet의 깊이별 레이어의 빈도로 인해 높으며, 디코더에서 깊이별 레이어를 사용하면 더욱 증가한다. 깊이별 레이어의 관측된 실행 시간 비효율성을 해결하기 위해 해당 논문은 TVM 컴파일러 스택을 사용한다. TVM은 하드웨어별 스케줄링과 연산자 튜닝을 수행하여 줄어든 연산의 영향을 줄어든 처리 시간으로 변환할 수 있게 한다. 표 6의 오른쪽 부분은 TVM으로 컴파일된 네트워크의 TX2 실행 시간을 보여준다. 디코더에서의 깊이 분해는 이제 CPU 실행 시간을 3.5배로, GPU 실행 시간을 2.5배로 줄인다.
Network Pruning 이전에는 해당 논문의 아키텍처(MobileNet-NNConv5, 디코더에서의 깊이별 분해, 가산 스킵 연결)가 이미 TX2 GPU에서 실시간 처리량을 초과하지만, 아직 TX2 CPU에서 실시간 속도를 달성하지 못했다. network pruning은 모델의 실행 시간을 낮춰 CPU 프레임레이트를 25 fps 이상으로 높일 수 있어 실시간 추론에 더 적합하다.

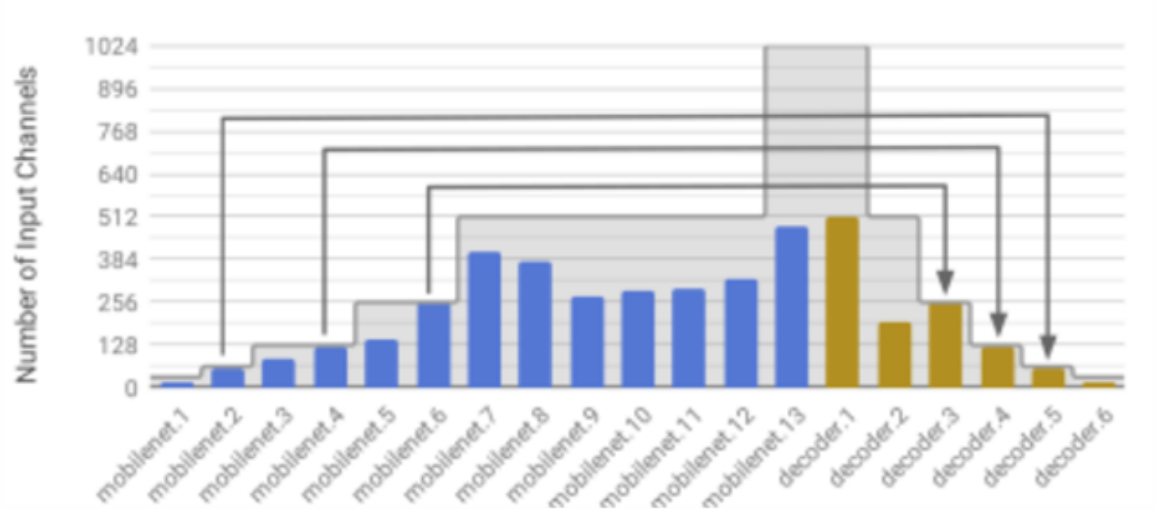
<표 6: 하드웨어별 실행 시간>

MobileNet-NNConv5	in PyTorch		using TVM	
	CPU [ms]	GPU [ms]	CPU [ms]	GPU [ms]
with standard decoder	**4100**	34.9	176	20.9
with depthwise decoder	5200	**18.6**	**50**	8.3
with depthwise & skip-add	5100	19.1	66	**8.2**

표 7에 나와 있는 것처럼, pruning은 MACs를 2배로 줄이고, GPU 실행 시간을 1.5배로 줄이며, 거의 동일한 정확도로 GPU 실행 시간을 1.8배로 줄인다. 그림 4(e)에서는 pruning 과정이 출력 깊이 맵의 선명도와 시각적 명확성을 보존한다는 것을 보여준다. 그림 6에서는 pruning된 아키텍처를 보여준다. 인코더에서 하나 (레이어 mobilenet.9)와 디코더에서 하나 (레이어 decoder.2)의 병목 현상이 있는 것을 볼 수 있다.

	Before Pruning	After Pruning	Reduction
Weights	3.93M	1.34M	2.9×
MACs	0.74G	0.37G	2.0×
RMSE	0.599	0.604	-
δ_1	0.775	0.771	-
CPU [ms]	66	37	1.8×
GPU [ms]	8.2	5.6	1.5×

<표 7: 인코더-디코더 network pruning 성능>



<그림 6: pruning 후 네트워크 아키텍처의 각 레이어에 대한 입력 채널 수>

해당 논문에서는 임베디드 시스템에서 고속 깊이 추정을 가능하게 한다. 효율적인 네트워크 아키텍처를 개발하여 낮은 복잡성 및 낮은 지연 디코더 설계로 높은 프레임 속도를 달성한다. 이 설계는 작은 MobileNet 인코더와 결합되더라도 추론 런타임을 영향을 받지 않는다. 제안된 모델의 소형 모델 크기는 최신 Pruning 알고리즘을 적용함으로써 더욱 축소된다. 하드웨어별 컴파일레이션을 사용하여 복잡성 감소를 대상 플랫폼에서 낮은 런타임으로 변환한다. Jetson TX2에서 최종 모델은 이전 연구보다 한 차원 빠른 런타임을 달성하면서 비슷한 정확도를 유지한다. 해당 논문은 깊이 추정에 중점을 두지만, 비슷한 접근법이 이미지 분할과 같은 다른 밀집 예측 작업을 위해 실시간 성능을 달성하는 데 사용될 수 있다고 생각한다.