

Università degli Studi di Padova
Dipartimento di Matematica "Tullio Levi-Civita"
Corso di Laurea in Informatica

Predizione della Profondità con Deep Learning da Immagini di Telecamera Monoculare

Esame di laurea

Indice

- *Monocular Depth Estimation*
- PyDNet e migrazione
- XiNet e l'attention
- Ricerca e sviluppo

Metodi tradizionali di misurazione delle profondità

- Sensori di profondità (e.g. LiDAR)
 - ✓ Facili da usare
 - ✗ Spesso molto costosi
- Stereocamere con algoritmi di calcolo delle disparità
 - ✗ Difficili da usare
 - ✓ Poco costose (rispetto ai sensori)

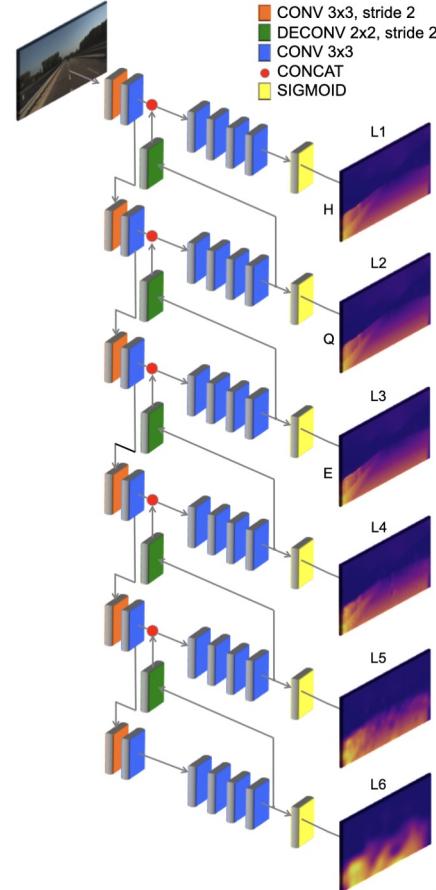
Cosa si spera di ottenere?

- Il costo delle fotocamere
- La praticità dei sensori

Deve essere adatto per un sistema *embedded*

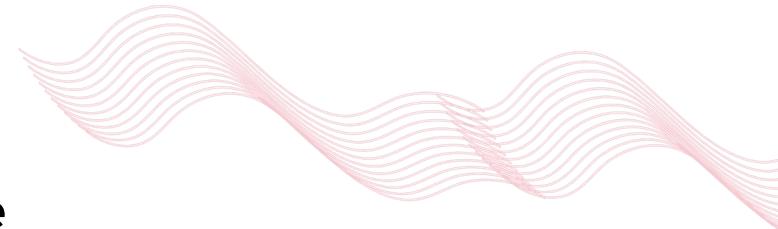
- Poca memoria
- Poca energia
- Poca potenza computazionale

Il modello



[paper](#) (VI)

PyDNet e migrazione



✓ Efficiente

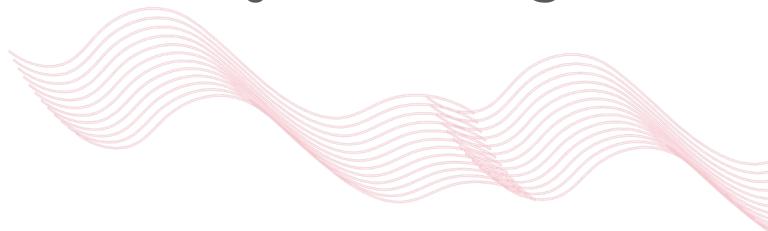
- Modello molto piccolo
- Sfrutta solo operazioni lineari

✓ Buona efficacia

- Buoni risultati qualitativi
- Risultati quantitativi vicini allo stato dell'arte

✗ Implementato in TensorFlow 1.8

- Versione deprecata del framework
- Incompatibile con versioni di CUDA attuali



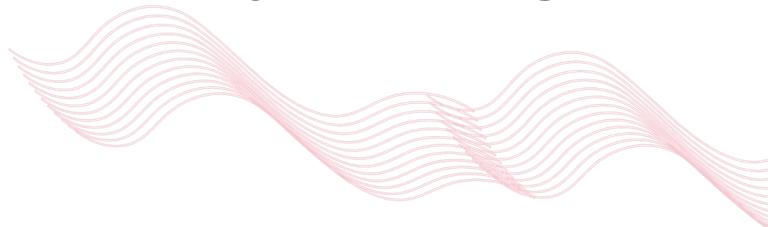
Dataset

- Unità che approvvigiona i dati
- **Reso più veloce** sfruttando:
 - Pandas: lettura dei file testuali
 - Pillow: lettura delle immagini
- LOC: 217



Modelli

- Modelli effettivi da allenare
- **Reso più leggibile** sfruttando:
 - `torch.nn.Module`
 - `torch.nn.Sequential`
- LOC: 362



Configurazione

- Unità che configura modelli e procedure
- **Reso più comodo** sostituendo comandi da terminale con oggetti di configurazione
- LOC: 303

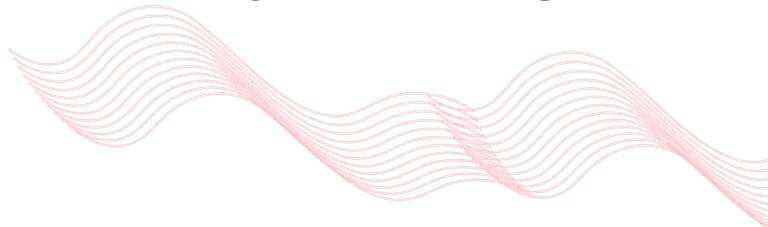


Allenamento

- Procedura di allenamento
- **Migliorata la leggibilità** delle *loss function*
- **Migliorato il salvataggio** dei *checkpoint*
- LOC: 518

La migrazione (3/3)

PyDNet e migrazione



Utilizzo

- Procedura di utilizzo del modello
- Migrato l'utilizzo mediante terminale
- Migrato l'utilizzo mediante *live webcam*
- **Implementato l'utilizzo mediante modulo importabile**
- LOC: 370



Valutazione

- Procedura di *testing* e validazione
- Riscrittura della procedura di *testing retrocompatibile* con la procedura di validazione originale
- LOC: 134

I risultati

PyDNet e migrazione



KITTY 50

Fonte	Minore è meglio				Maggiore è meglio		
	Abs Rel	Sq Rel	RMSE	RMSE log	d1	d2	d3
PDV1	<u>0.163</u>	1.399	<u>6.253</u>	<u>0.262</u>	<u>0.759</u>	<u>0.911</u>	<u>0.961</u>
PDV1 in PyTorch	0.16	1.52	6.229	0.253	0.782	0.916	0.964



CS+KITTY 50

Fonte	Minore è meglio				Maggiore è meglio		
	Abs Rel	Sq Rel	RMSE	RMSE log	d1	d2	d3
PDV1	<u>0.148</u>	1.318	<u>5.932</u>	<u>0.244</u>	<u>0.8</u>	<u>0.925</u>	<u>0.967</u>
PDV1 in PyTorch	0.147	1.378	5.91	0.242	0.804	0.927	0.967



KITTY 200

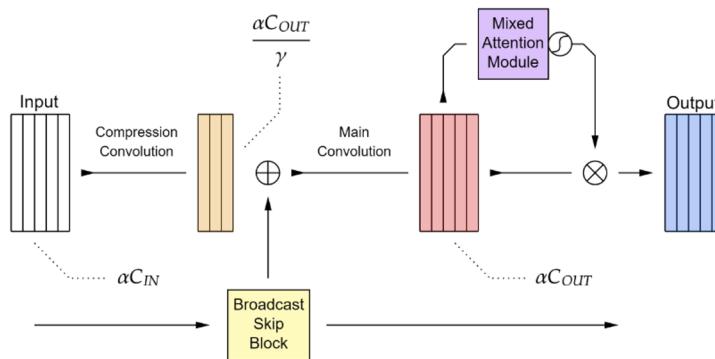
Fonte	Minore è meglio				Maggiore è meglio		
	Abs Rel	Sq Rel	RMSE	RMSE log	d1	d2	d3
PDV1	0.153	1.363	6.03	<u>0.252</u>	0.789	0.918	<u>0.963</u>
PDV1 in PyTorch	0.153	1.473	6.23	0.251	0.789	0.918	0.964

Esperimenti provati:

- *Dataset in bianco e nero* ✗
- *Dataset in formato HSV* ✗
- *Data augmentation ribaltamento verticale* ✗
- *Data augmentation solo ribaltamento verticale* ✗
- Aumentare risoluzione immagini in input ✓

Fonte	Minore è meglio				Maggiore è meglio		
	Abs Rel	Sq Rel	RMSE	RMSE log	d1	d2	d3
PDV1	<u>0.16</u>	<u>1.52</u>	<u>6.229</u>	<u>0.253</u>	<u>0.782</u>	<u>0.916</u>	<u>0.964</u>
PDV2	0.157	1.487	6.167	0.254	0.783	0.917	0.964

[paper](#)



XiConv

- Riduce l'impatto energetico rispetto alle convoluzioni
- Ottimi *benchmark* di valutazione
- Progettato per l'uso in sistemi *embedded*
- [paper](#)



Self attention

- Complessità quadratica nella dimensione dell'input
- Pesa ogni “pixel” usando le correlazioni con tutti gli altri
- Molto costoso ma molto efficace
- [paper](#)

Convolutional Block Attention Module

- Complessità lineare nella dimensione dell'input
- Sfrutta convoluzioni e pooling
- Meno costoso ma discretamente efficace
- [paper](#)

Modelli implementati a confronto

Ricerca e sviluppo

Fonte	Minore è meglio							Maggiore è meglio		
	#p	Inf. (s)	Abs	Sq Rel	RMSE	RMSE log	d1	d2	d3	
			Rel							
PDV1	1971624	0.15	0.16	1.52	6.229	0.253	0.782	0.916	0.964	
PDV2	716680	0.1	0.157	1.487	6.167	0.254	0.783	0.917	0.964	
PyXiNet α I	429661	0.14	0.17	1.632	6.412	0.269	0.757	0.903	0.958	
PyXiNet α II	709885	0.12	0.168	1.684	6.243	0.259	0.777	0.913	0.96	
PyXiNet β I	941638	0.16	0.156	1.546	6.259	0.251	0.791	0.921	0.965	
PyXiNet β II	481654	0.14	0.168	1.558	6.327	0.259	0.762	0.91	0.963	
PyXiNet β III	1246422	0.16	0.148	1.442	6.093	0.241	0.803	0.926	0.967	
PyXiNet β IV	1446014	0.18	0.146	1.433	6.161	0.241	0.802	0.926	0.967	
PyXiNet M I	1970643	0.36	0.147	1.351	5.98	0.244	0.8	0.926	0.967	
1	PyXiNet M II	2233197	0.38	0.14	1.289	5.771	0.234	0.814	0.933	0.969
PyXiNet M III	1708499	0.35	0.141	1.279	5.851	0.239	0.808	0.927	0.968	
PyXiNet M IV	1839981	0.36	0.145	1.25	5.885	0.242	0.798	0.926	0.967	
2	PyXiNet β CBAM I	1250797	0.19	0.143	1.296	5.91	0.239	0.805	0.928	0.968
PyXiNet β CBAM II	1450389	0.23	0.147	1.379	5.974	0.239	0.806	0.927	0.968	
CBAM PyDNet	746673	0.28	0.167	1.722	6.509	0.251	0.776	0.916	0.965	

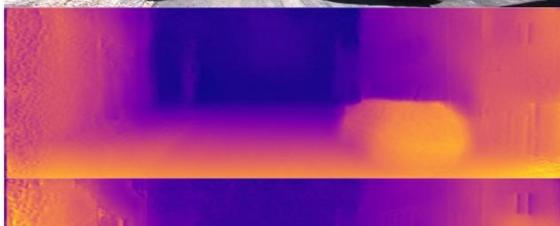
Confronti qualitativi

Ricerca e sviluppo

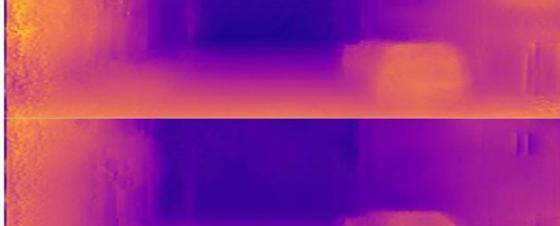
Input image



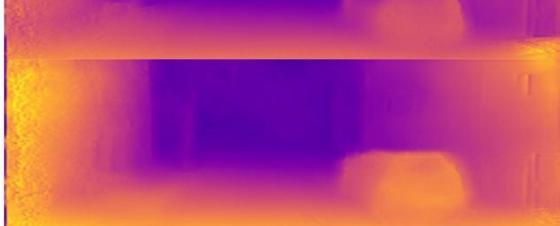
PyDNet V1



PyDNet V2



PyXiNet M II



PyXiNet β CBAM I



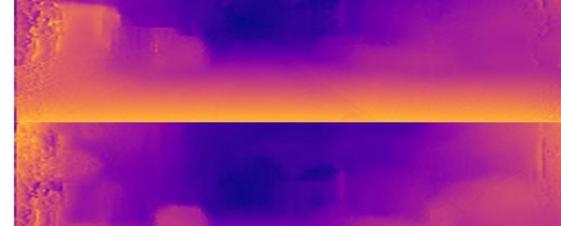
Input image



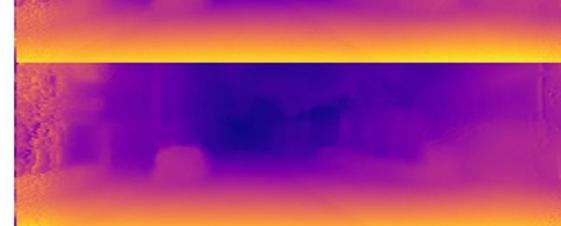
PyDNet V1



PyDNet V2



PyXiNet M II



PyXiNet β CBAM I



Report quantitativo

320 ore di lavoro

- 80 ore di studio (*framework e paper*)
- 80 ore migrazione PyDNet
- 40 ore di sperimentazione con PyDNet
- 120 ore di *R&D*

3906 LOC

- 1904 (48%) in migrazione
- 2002 (52%) in *R&D*

15 modelli implementati

- PyDNet V1 & V2
- 13 modelli sperimentali

PyXiNet β CBAM I vs. PyDNet V1

- Miglioramenti minimo 1%
- Miglioramento massimo 15%
- 37% di parametri in meno
- *Inference time* vicini