PVNet: Pixel-wise Voting Network for 6DoF Pose Estimation

2019/6/30 neka-nat

自己紹介

- 名前：neka-nat
- 職業：とある製造メーカのソフトウェアエンジニア
- 普段のお仕事
  - 画像処理やロボットのソフト開発
- 最近の興味：点群処理、CG
- https://twitter.com/neka_nat

この論文を選んだ理由

- Oralの論文から選択
- 昔からある問題設定だが、深層学習によってかなり精度が上がってきている
- LINEMODデータセットでSOTA
- 本日の内容
  - 問題設定について
  - 関連研究について
  - PVNetについて

論文の問題設定

- 物体の種類と位置姿勢推定を同時に行いたい
  - 物体の種類
  - 物体の位置(XYZの3次元)
  - 物体の姿勢(ロールピッチヨーの3次元)

} 合わせて6次元

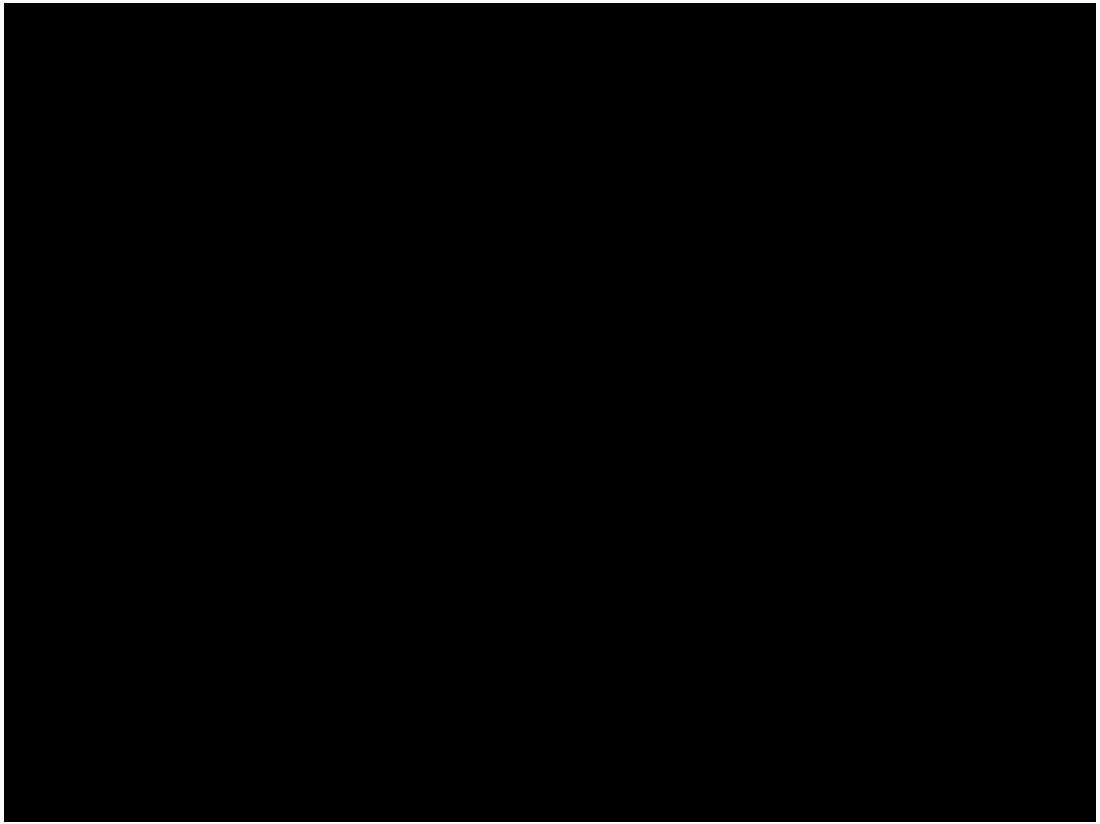
- 1枚のRGB画像のみを入力とする
- 物体のCADモデルを使用できる




応用例

- ロボットビジョン
- AR
- デプスセンサなどが使用しにくい環境・状況
  - 小型化
  - 屋外での使用
  - コスト削減

AR demo



よく用いられるデータセット

- それぞれCADモデルとRGBD画像が用意されている
- LINEMOD, Occlusion LINEMOD
  - 15種類の一般物体、オクルージョンがある・無いデータが分かれている
- T-LESS
  - 30種類の産業物体、テクスチャが無く似た物体が多い
- YCB
  - 5つにカテゴリ分けされた77の一般物体

評価メトリクス

- 6D Pose (ADD)
  - 以下の式の値が物体の直径の10%以内、xは3Dモデル上の点

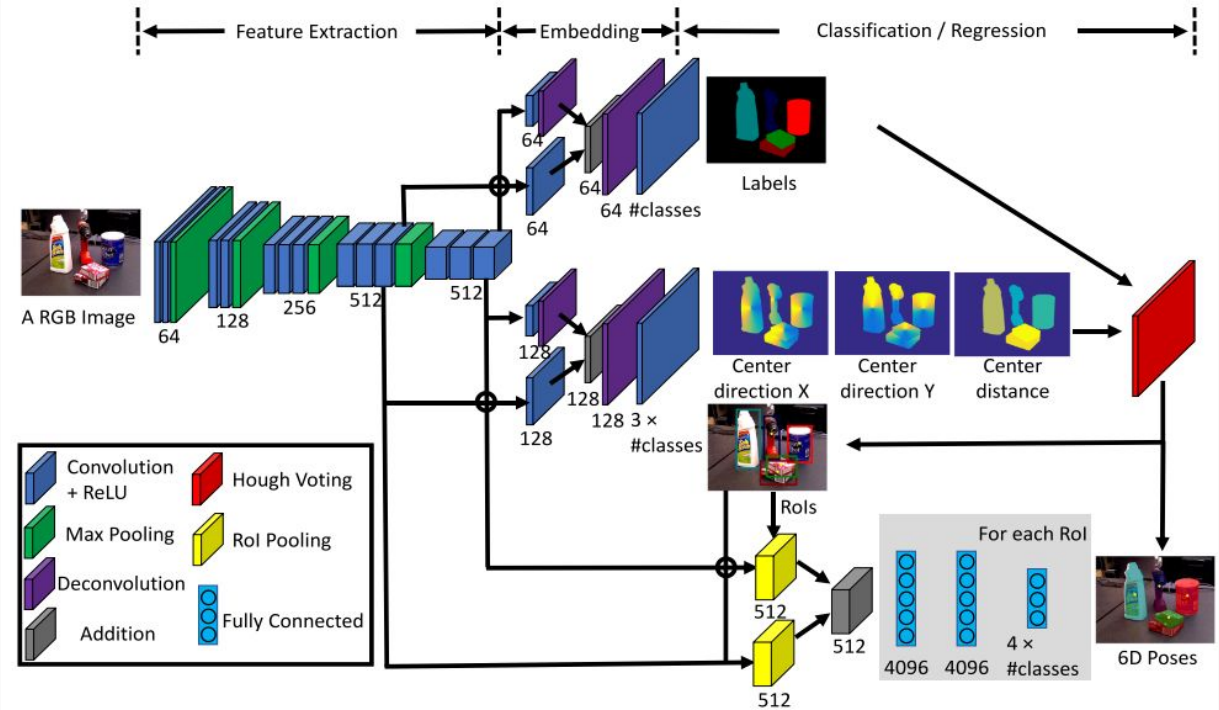
$$\mathrm{ADD} = \frac{1}{m} \sum_{\mathbf{x} \in \mathcal{M}} \left\| (\mathbf{R}\mathbf{x} + \mathbf{T}) - (\tilde{\mathbf{R}}\mathbf{x} + \tilde{\mathbf{T}}) \right\|$$

正解の並進と回転　　推定した並進と回転

- Projection 2D
  - 2次元に投影したエラーに対してしきい値を設定

関連論文①End-to-Endでやってみたーその1

- PoseCNN(RSS2018)
- ラベル、3次元位置、3次元姿勢の3つの出力を持つネットワーク



ラベル

3次元位置

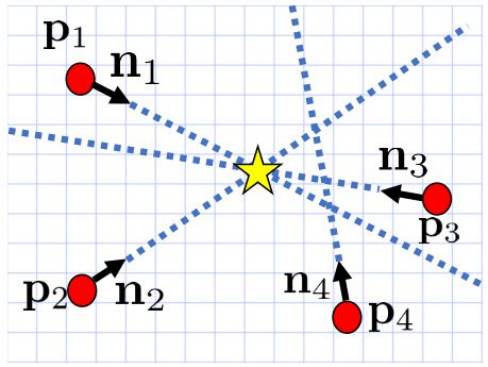
物体中心に向かうベクトル場を推定し、Votingによって物体中心を求める

3次元姿勢

物体中心の推定結果と合わせてクォータニオンを出力する

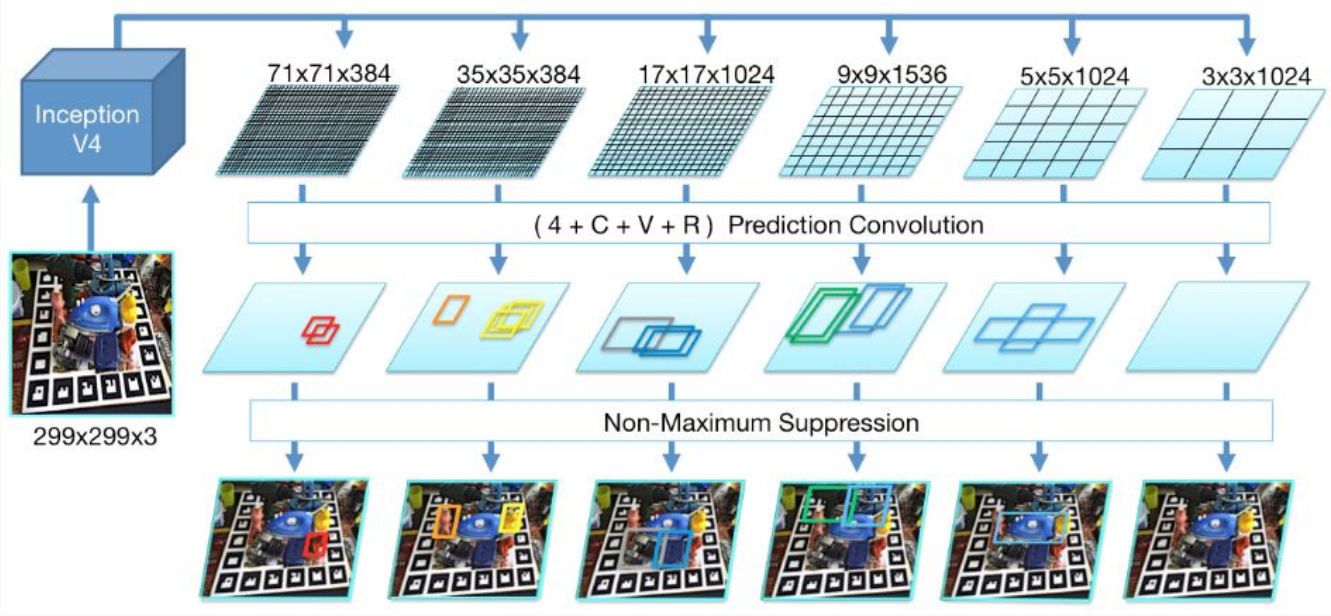
関連論文①End-to-Endでやってみたーその1

- 物体の3次元位置を指すベクトル場を学習することでオクルージョンに強い
- 位置はロバストに推定できたが、姿勢の推定が弱い



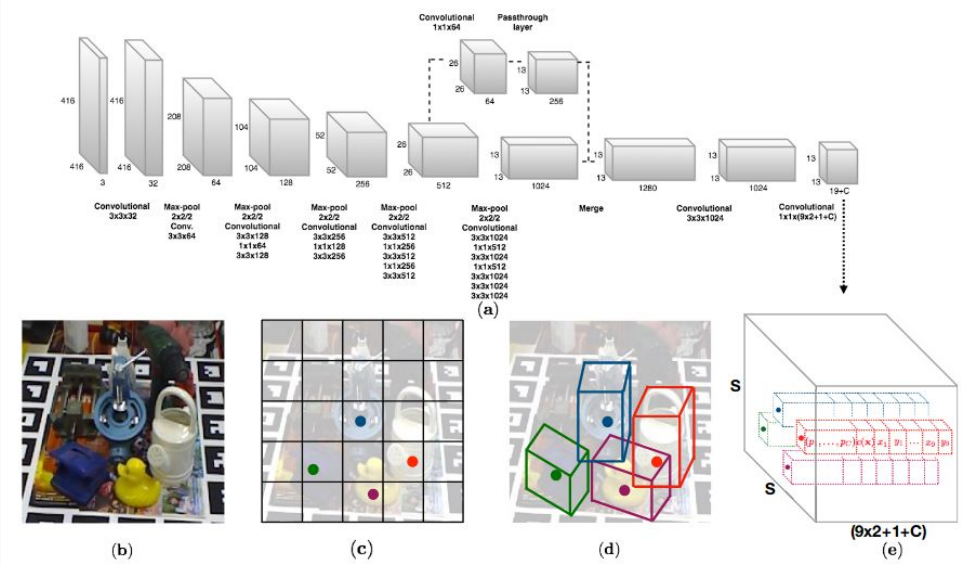
関連論文②End-to-Endでやってみたーその2

- SSD-6D(ICCV2017)
- SSDをベースに姿勢推定を5°刻みのクラスタリング問題にして解く
  - よりロバストに姿勢が求まるが精度が落ちる
- 出力をさらにRGB画像を用いたエッジベースのRefinementにかける

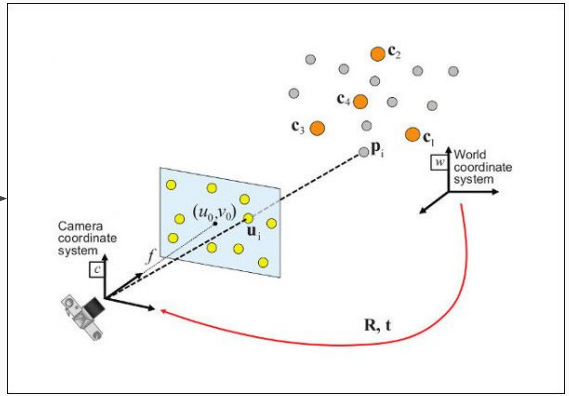


関連論文③2次元の特徴点を抽出してからのPnPを解く

- Tekinらの手法(CVPR2018)
- YOLOをベースにして高速化、50fps(SSD-6Dの5倍高速化)
- 問題を2つに分ける
  - 物体の境界BOXを2次元投影した頂点(8点)＋中心点を画像から推定
  - 2次元投影した境界BOXの対応点からPnP(Perspective-n-Points)によって位置姿勢を求める



PnP



関連論文④Refinementもディープにしてみた

- BB8(ICCV2017)
- 特徴点抽出(境界BOXの頂点)＋PnP＋Refinementの構成
- RefinementにCNNを用いた

物体周辺を切り
取った入力画像

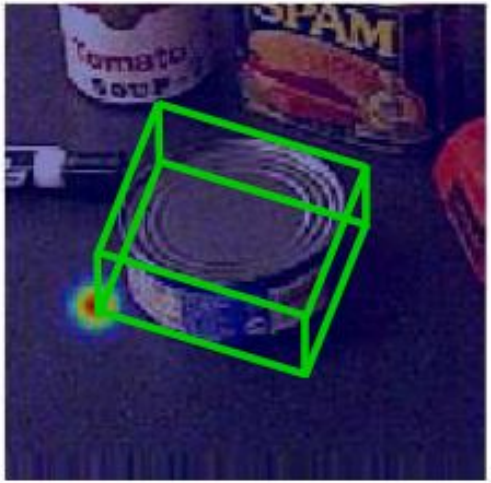
CADモデルを初期姿
勢でレンダリング

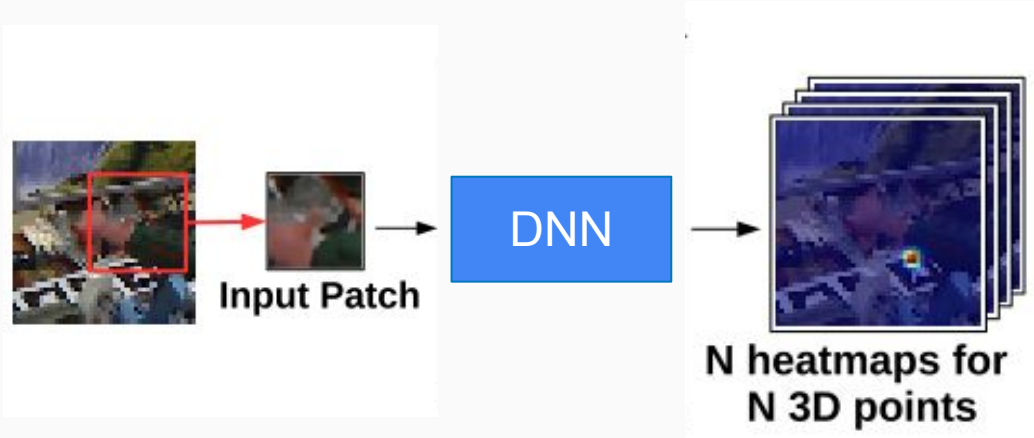
CNN

境界BOXの頂点の移動量

関連論文⑤オクルージョンに強いモデルの構築

- Oberwegerらの手法(ECCV2018)
- オクルージョンへの対応を意識した手法
- 入力画像のパッチから特徴点のヒートマップを出力
- 各パッチの出力を累積して特徴点を推定する



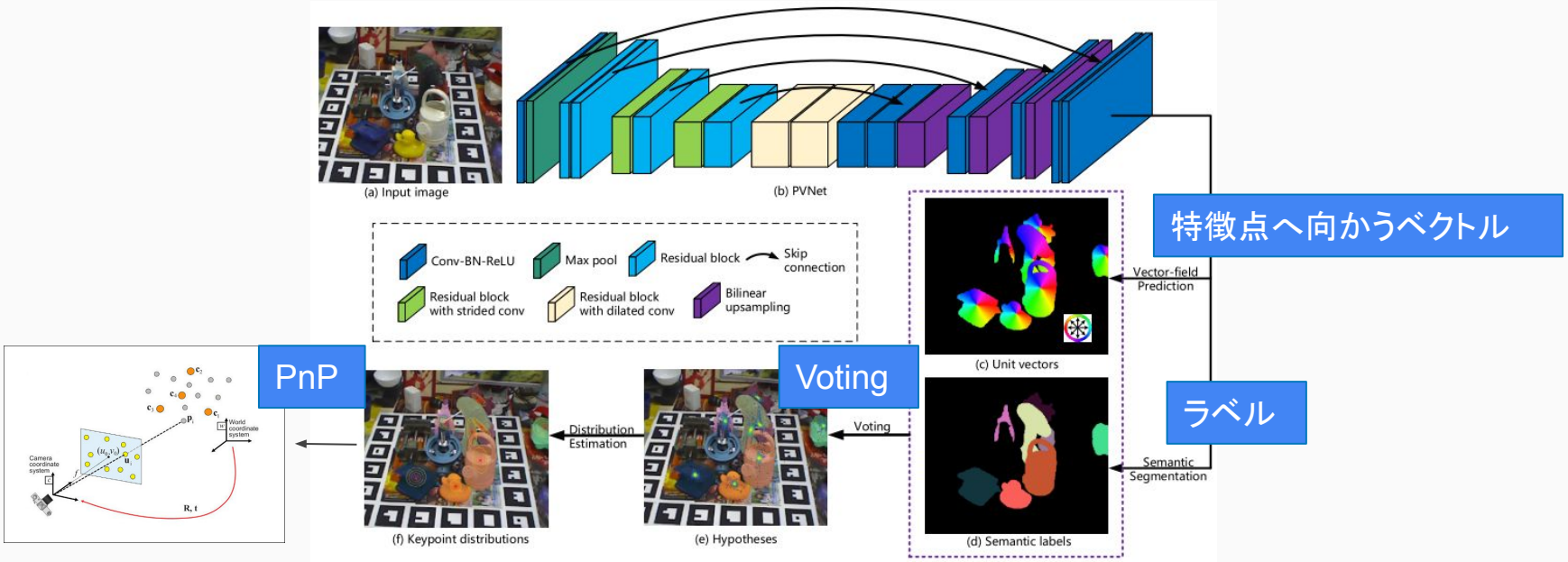


PVNet

- 関連研究で分かったこと
  - PoseCNNのようなベクトル場からのVotingはオクルージョンに強い
  - 特徴点推定→PnPの2段階にするのが良さそうだが、境界BOXを使った物体の外にある特徴点はイマイチ
  - Refinementは有効だが、物体毎に計算が必要になり計算時間が増加する

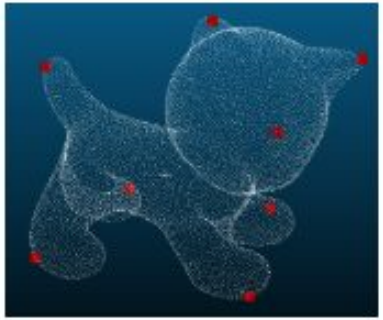
PVNet

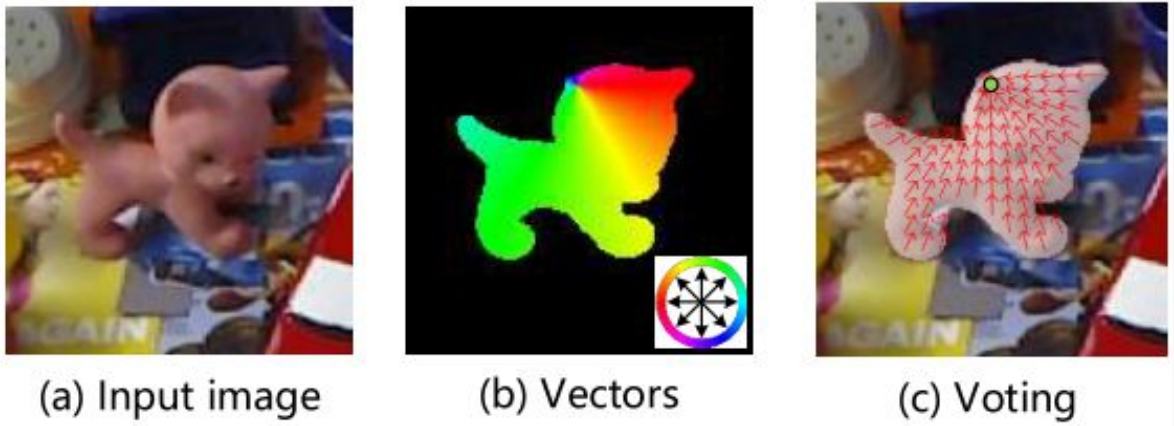
- 関連研究を踏まえての方針
  - 2次元の特徴点推定→PnPの2段階で解く
  - 計算量の観点からRefinementは行わないで解きたい
  - 特徴点推定にベクトル場の推定とVotingを取り入れる(ロバスト性向上)
  - 推定された特徴点の不確実性を考慮してPnPを解く(精度向上)



特徴点推定

- Farthest-Point-Samplingを用いて事前にCADモデル上の8点の特徴点と中心点を求める
- RGB画像から各特徴点＋中心点の方向を示すベクトル場を推定する
- 各ピクセルの出力次元は「(特徴点＋中心点)×ベクトルの次元+カテゴリ数」





(a) Input image　(b) Vectors　(c) Voting

Uncertainty-driven PnP

- RANSACベースのVotingによって特徴点の候補点**h**をいくつ求める
- 候補点を特徴点とした場合のベクトル場と推定されたベクトル場との一致度合いを候補点のスコアとする
- 候補点とそのスコアから特徴点の平均値と分散を求める

候補点の重み

$$w_{k,i} = \sum_{\mathbf{p} \in O} \mathbb{I}\left(\frac{(\mathbf{h}_{k,i} - \mathbf{p})^T}{\|\mathbf{h}_{k,i} - \mathbf{p}\|_2} \mathbf{v}_k(\mathbf{p}) \geq \theta \right)$$

特徴点の推定値

$$\boldsymbol{\mu}_k = \frac{\sum_{i=1}^{N} w_{k,i} \mathbf{h}_{k,i}}{\sum_{i=1}^{N} w_{k,i}},$$

推定値の分散

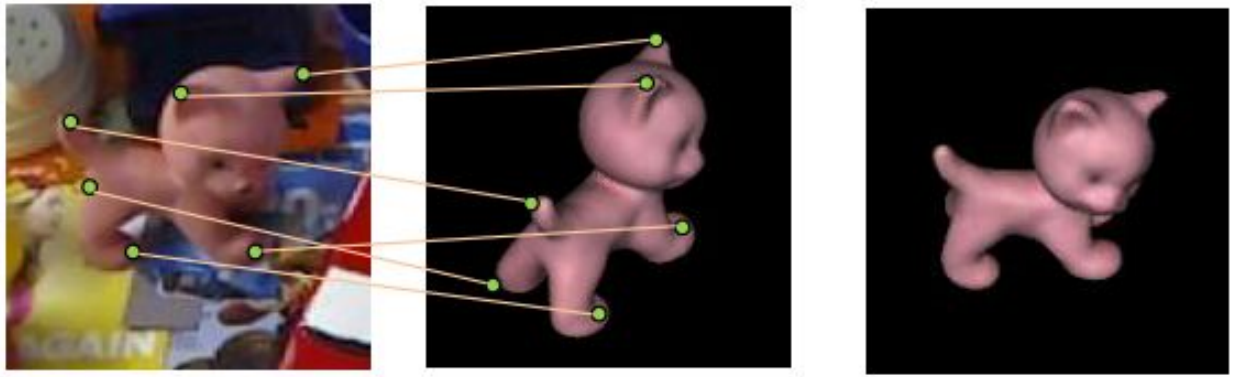
$$\boldsymbol{\Sigma}_k = \frac{\sum_{i=1}^{N} w_{k,i} (\mathbf{h}_{k,i} - \boldsymbol{\mu}_k)(\mathbf{h}_{k,i} - \boldsymbol{\mu}_k)^T}{\sum_{i=1}^{N} w_{k,i}},$$

Uncertainty-driven PnP

- 推定された特徴点と分散の両方を考慮したPnP問題を解く

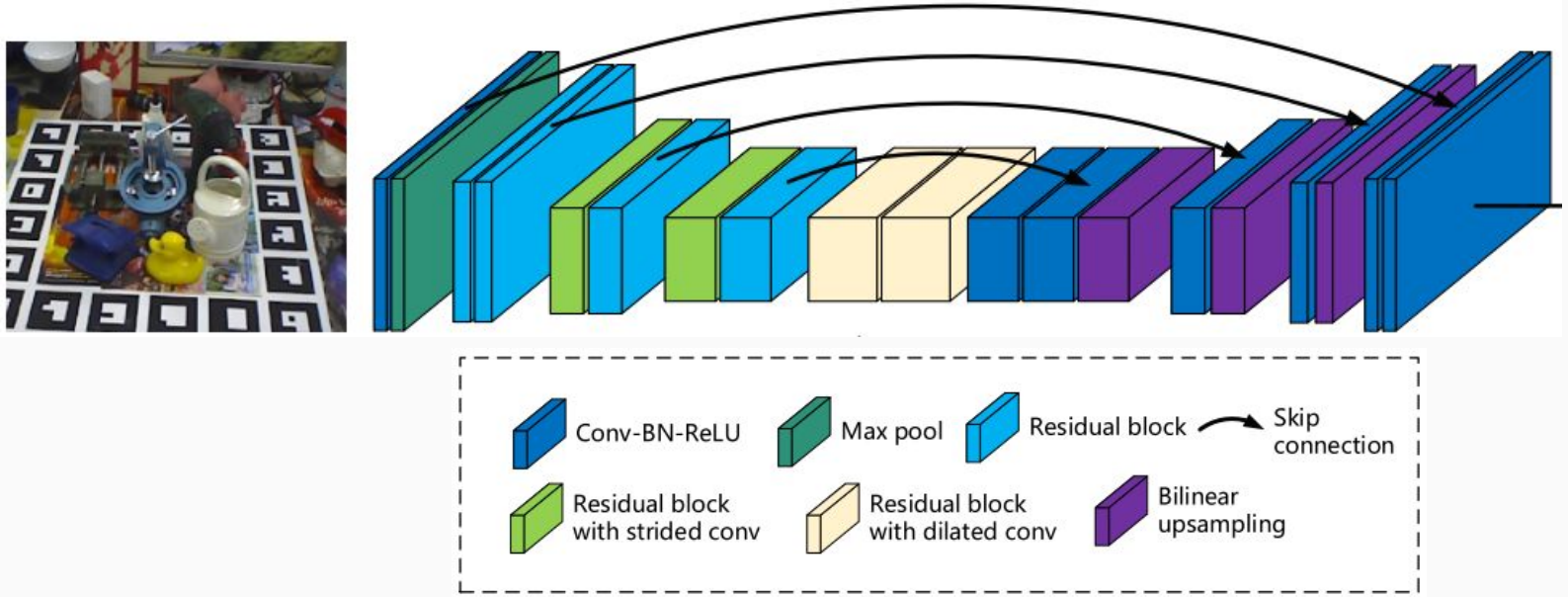
$$\underset{R,\mathbf{t}}{\text{minimize}} \sum_{k=1}^{K} (\tilde{\mathbf{x}}_k - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} (\tilde{\mathbf{x}}_k - \boldsymbol{\mu}_k),$$

$$\tilde{\mathbf{x}}_k = \pi(R\mathbf{X}_k + \mathbf{t}),$$



ネットワークアーキテクチャ

- Pretrained ResNet18をバックボーンにしている
- 中間画像のサイズがH/8×W/8になった時点でPoolingを行わない
- 中間画像のサイズを維持しつつ情報集約を行うためにDilated Convを使用
- スキップするところはStrideで調整



学習方法

- 正解のベクトル場と推定結果を用いてSmooth L1 lossを計算
- 過学習を防ぐため合成画像を20000枚追加

$$\ell(\mathbf{w}) = \sum_{k=1}^{K} \sum_{\mathbf{p} \in O} \ell_1(\Delta \mathbf{v}_k(\mathbf{p}; \mathbf{w})|_x) + \ell_1(\Delta \mathbf{v}_k(\mathbf{p}; \mathbf{w})|_y),$$

$$\Delta \mathbf{v}_k(\mathbf{p}; \mathbf{w}) = \tilde{\mathbf{v}}_k(\mathbf{p}; \mathbf{w}) - \mathbf{v}_k(\mathbf{p}), \qquad (6)$$

Ablation study

- Occlusion LINEMODで様々に構成要素を変えて比較
- 特徴点の数を4、8、12で変化させて比較
- Uncertainty-driven PnPと普通のPnP(EPnP)との比較

methods	Tekin [36]	BBox 8	Offset 8	FPS 4	FPS 8	FPS 12	FPS 8 + Un
ape	2.48	6.50	12.99	5.31	**17.44**	15.1	15.81
can	17.48	65.04	**69.10**	18.81	63.21	64.87	63.30
cat	0.67	15.00	**26.12**	16.01	17.35	16.68	16.68
duck	1.14	15.95	14.55	13.85	**26.12**	24.89	25.24
driller	7.66	55.60	65.24	12.19	62.19	64.17	**65.65**
eggbox	-	35.23	41.62	36.77	44.96	41.53	**50.17**
glue	10.08	42.64	**55.48**	24.81	47.32	51.94	49.62
holepuncher	5.45	35.06	32.22	15.98	39.50	**40.16**	39.67
average	6.42	33.88	39.66	17.96	39.76	39.92	**40.77**

特徴点を境界BOXにしてみた

ベクトル場でなくオフセットを使用

特徴点の数をそれぞれ変更

Uncertainty-driven PnPを使用

実験結果

- LINEMODによるADDを用いた従来手法との比較
- Tekinらの方法から30%以上精度が向上している
- Refinementを行ったSSD-6Dよりも精度が良い

Refinement無し

Refinement有り

methods	w/o refinement				w/ refinement	
	BB8 [33]	SSD-6D [20]	Tekin [39]	OURS	BB8 [33]	SSD-6D [20]
ape	27.9	0.00	21.62	43.62	40.4	**65**
benchwise	62.0	0.18	81.80	**99.90**	91.8	80
cam	40.1	0.41	36.57	**86.86**	55.7	78
can	48.1	1.35	68.80	**95.47**	64.1	86
cat	45.2	0.51	41.82	**79.34**	62.6	70
driller	58.6	2.58	63.51	**96.43**	74.4	73
duck	32.8	0.00	27.23	52.58	44.30	**66**
eggbox	40.0	8.90	69.58	99.15	57.8	**100**
glue	27.0	0.00	80.02	95.66	41.2	**100**
holepuncher	42.4	0.30	42.63	**81.92**	67.20	49
iron	67.0	8.86	74.97	**98.88**	84.7	78
lamp	39.9	8.20	71.11	**99.33**	76.5	73
phone	35.2	0.18	47.74	**92.41**	54.0	79
average	43.6	2.42	55.95	**86.27**	62.7	79

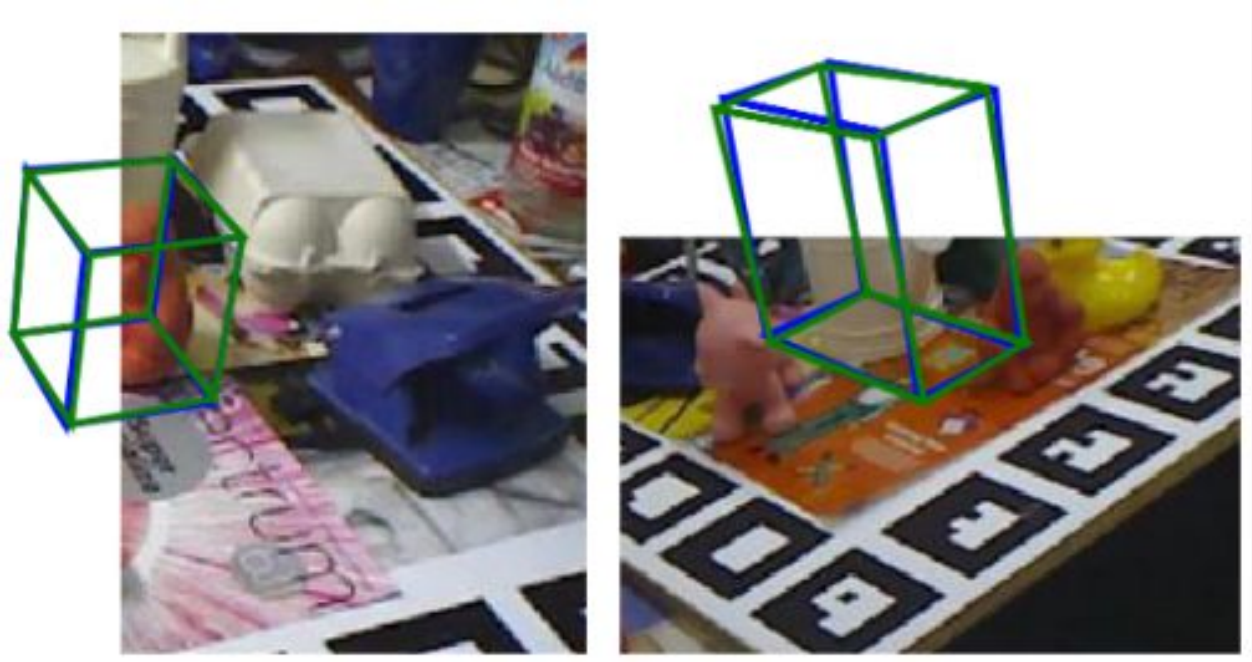
実験結果

- Occlusion LINEMODによるADDを用いた従来手法との比較
- Oberwegerらの手法よりも10%以上精度が向上している

methods	Tekin [36]	PoseCNN [40]	Oberweger [27]	OURS
ape	2.48	9.6	**17.6**	15.81
can	17.48	45.2	53.9	**63.30**
cat	0.67	0.93	3.31	**16.68**
duck	1.14	19.6	19.2	**25.24**
driller	7.66	41.4	62.4	**65.65**
eggbox	-	22	25.9	**50.17**
glue	10.08	38.5	39.6	**49.62**
holepuncher	5.45	22.1	21.3	**39.67**
average	6.42	24.9	30.4	**40.77**

実験結果

- Truncation LINEMOD(著者らが作成した端が切れた画像)に対しも実験を行っており、ADDで平均31.48%の精度を出している



その他の実験結果

- YCBデータセットでも他手法と比較した
- 計算速度は480×640の画像で25fps
- Intel i7 3.7 CPU, GTX1080Tiを使用

methods	PoseCNN [40]	Oberweger [27]	OURS
2D Projection	3.72	39.4	**47.4**
ADD(-S) AUC	61.0	72.8	**73.4**

まとめ

- CVPR2019で発表された6次元姿勢推定手法PVNetを紹介した
- Refinement無しで高精度に推定可能
- Occlusion, truncationにも対応できる
- 著者によるPytorchのソースコードが利用できる
  - https://github.com/zju3dv/pvnet
- トレーニング画像生成
  - https://github.com/zju3dv/pvnet-rendering

おわり