Lab: Place Localization with PoseNet

SUNY Korea - Francois Rameau

ANAIS 2023

Content

1. *Introduction*
2. *Training: Network design and data loader*
3. *Localization error metric*
4. *Analyze corner cases*
5. *Robustness to noise*
6. *Test on unseen images*

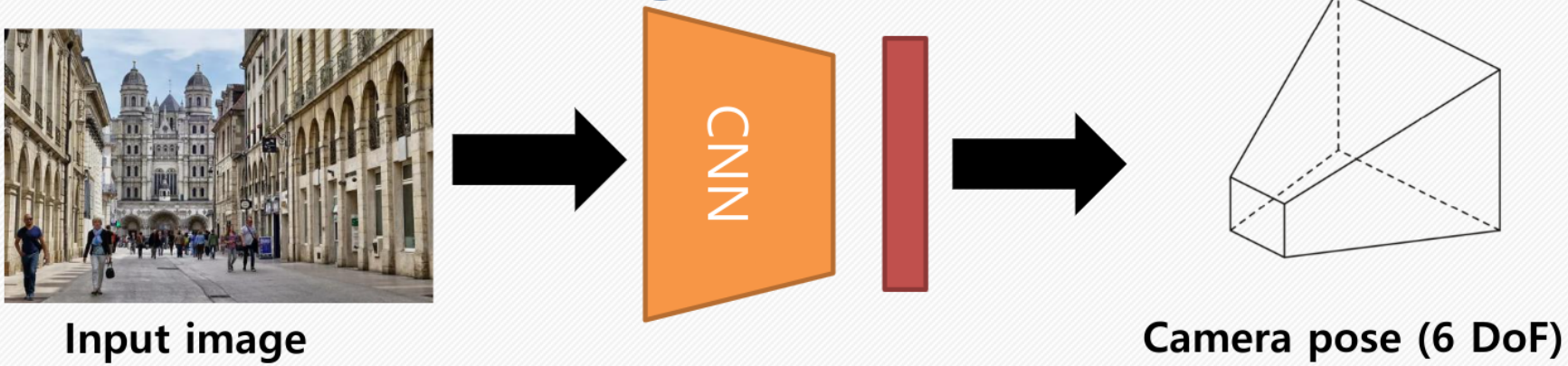
colab

LINK TO THE LAB

I – PoseNet: Recap

PoseNet : **Supervised** Learning for **camera pose** estimation.

PoseNet in one image



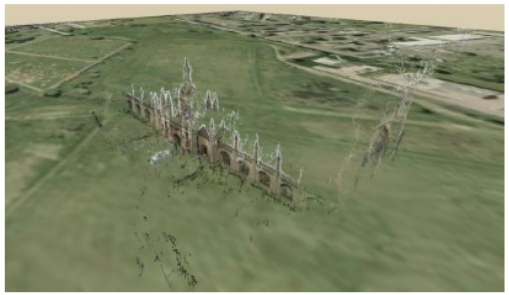
Reference: Kendall, Alex, Matthew Grimes, and Roberto Cipolla. "Posenet: A convolutional network for real-time 6-dof camera relocalization." *Proceedings of the IEEE international conference on computer vision*. 2015.

I – PoseNet: Dataset

King's College scene from Cambridge Landmarks


Training	Testing
1220 image-pose pairs	343 image-pose pairs

Structure-from-motion is used to create the labeled data

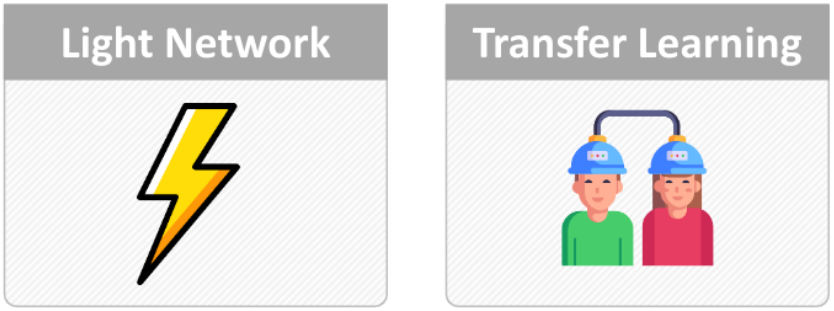


I – PoseNet: The network

Unlike the original implementation, we will use a lighter network

```
class PoseResNet(nn.Module):
    def __init__(self, num_features=2048, dropout=0.5):
        super(PoseResNet, self).__init__()
        self.base = models.resnet18(pretrained=True)
```

Why using a pre-trained ResNet18 on ImageNet?



I – PoseNet: The network

Unlike the original implementation, we will use a lighter network

```
class PoseResNet(nn.Module):
    def __init__(self, num_features=2048, dropout=0.5):
        super(PoseResNet, self).__init__()
        self.base = models.resnet18(pretrained=True)
        fc_in_features = self.base.fc.in_features
        self.base.avgpool = nn.AdaptiveAvgPool2d(1)
```

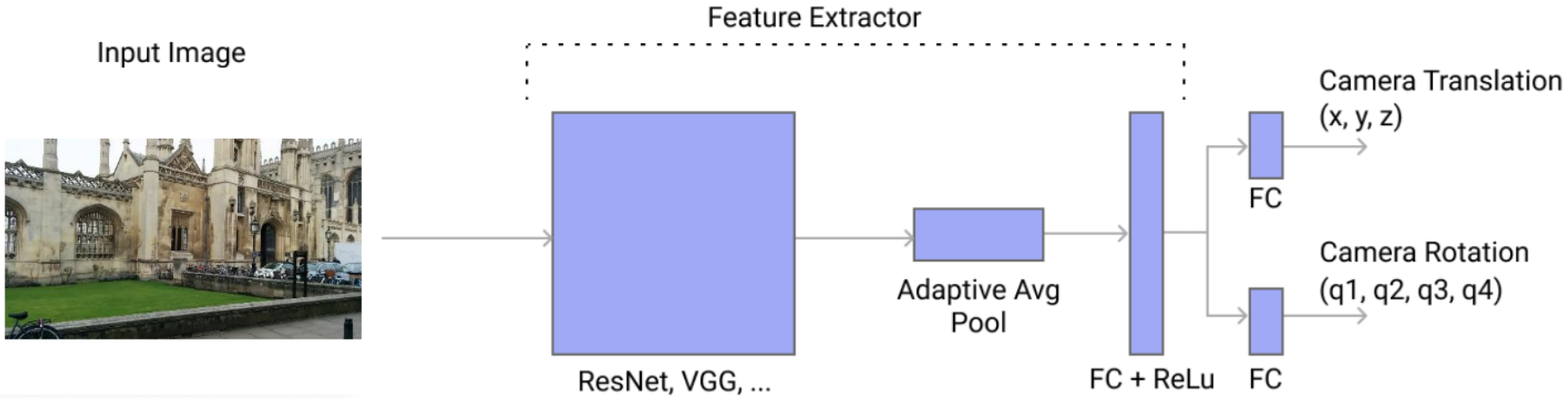
Adaptive average pooling (loosing spatial information but better global)

Adaptability	Dimensionality reduction	Preserve feature channels
Adaptive Average Pooling adjusts its operation to always produce an output of a specified size, making it flexible for inputs of varying sizes.	It condenses the spatial information of each feature map into a single value, simplifying the model's structure and reducing computational complexity.	While reducing spatial dimensions, Adaptive Average Pooling maintains the depth of feature channels, retaining all feature information extracted by previous layers.

I – PoseNet: The network

```
class PoseResNet(nn.Module):
    def __init__(self, num_features=2048, dropout=0.5):
        super(PoseResNet, self).__init__()
        self.base = models.resnet18(pretrained=True)
        fc_in_features = self.base.fc.in_features
        self.base.avgpool = nn.AdaptiveAvgPool2d(1)
        self.base.fc = nn.Sequential(
            nn.Linear(fc_in_features, num_features),
            nn.ReLU()
        )
        self.fc_xyz = nn.Linear(num_features, 3)
        self.fc_quat = nn.Linear(num_features, 4)
        self.dropout = dropout
```

Fully connected layers for translation and rotation predictions

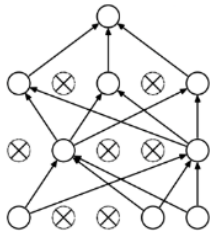


I – PoseNet: The network

```python
class PoseResNet(nn.Module):
    def __init__(self, num_features=2048, dropout=0.5):
        super(PoseResNet, self).__init__()
        self.base = models.resnet18(pretrained=True)
        fc_in_features = self.base.fc.in_features
        self.base.avgpool = nn.AdaptiveAvgPool2d(1)
        self.base.fc = nn.Sequential(
            nn.Linear(fc_in_features, num_features),
            nn.ReLU()
        )
        self.fc_xyz = nn.Linear(num_features, 3)
        self.fc_quat = nn.Linear(num_features, 4)
        self.dropout = dropout

    def extract_features(self, x):
        x_features = self.base(x)
        x_features = F.relu(x_features)
        if self.dropout > 0:
            x_features = F.dropout(x_features, p=self.dropout, training=self.training)
        return x_features
```

Dropout

Why is dropout important for this application? Better generalization to unseen pictures!



I – PoseNet: The network

```
class PoseResNet(nn.Module):
    def __init__(self, num_features=2048, dropout=0.5):
        super(PoseResNet, self).__init__()
        self.base = models.resnet18(pretrained=True)
        fc_in_features = self.base.fc.in_features
        self.base.avgpool = nn.AdaptiveAvgPool2d(1)
        self.base.fc = nn.Sequential(
            nn.Linear(fc_in_features, num_features),
            nn.ReLU()
        )
        self.fc_xyz = nn.Linear(num_features, 3)
        self.fc_quat = nn.Linear(num_features, 4)
        self.dropout = dropout

    def extract_features(self, x):
        x_features = self.base(x)
        x_features = F.relu(x_features)
        if self.dropout > 0:
            x_features = F.dropout(x_features, p=self.dropout, training=self.training)
        return x_features

    def forward(self, x):
        x_features = self.extract_features(x)
        x_translations = self.fc_xyz(x_features)
        x_rotations = self.fc_quat(x_features)
        x_poses = torch.cat((x_translations, x_rotations), dim=1)
        return x_poses
```

I – PoseNet: DataLoader

```python
class PoseDataset(Dataset):
    def __init__(self, root_dir, file_path, transform=None):
        self.root_dir = root_dir
        self.transform = transform
        self.data = []
        with open(file_path, 'r') as f:
            lines = f.readlines()[3:]  # skip the header
            for line in lines:
                items = line.split()
                image_path = os.path.join(root_dir, items[0])
                image = Image.open(image_path)
                if self.transform:
                    image = self.transform(image)
                pose = list(map(float, items[1:]))
                self.data.append((image, torch.tensor(pose)))

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        image, pose = self.data[idx]
        return image, pose

# Define the transformations
transform = transforms.Compose([
    transforms.Resize(260),
    transforms.CenterCrop(250),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225]),
])
```

Little trick:

To speed up the training time, the entire dataset is loaded in memory once.

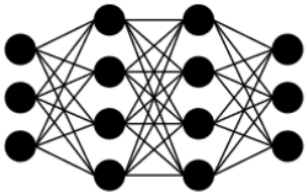
Only applicable for problem with small datasets

Data augmentation:

No data augmentation here, would you recommend some? Geometric transformations (rotation, translation)? Photometric transformation? Why or why not?c

I – PoseNet: The loss

- Pose (Position t and orientation R) Parametrization
  - $R \in \mathbb{R}^{3x3}, \boldsymbol{t} \in \mathbb{R}^3$



Rotation Matrix	Unit Quaternion e.g. [1]	Axis-Angle e.g. [2]	Log Unit Quaternion e.g. [3]
$R = \begin{pmatrix} r_1 & r_2 & r_3 \\ r_4 & r_5 & r_6 \\ r_7 & r_8 & r_9 \end{pmatrix}$	$\mathbf{q} = (c, v_1, v_2, v_3)$	$\log R = \theta\hat{\mathbf{u}} = (u_1', u_2', u_3')$	$\log \mathbf{q} = 2 \log R$ [4]
Enforce/Map: $RR^{\mathrm{T}} = I, \det(\mathrm{R}) = 1$	Enforce/Map: $\|\mathbf{q}\| = 1$	Enforce/Map: –	Enforce/Map: –

- Loss function:
  - Euclidean loss: $loss(I) = \|\hat{t} - t\|_2 + \beta \left\| \hat{q} - \frac{q}{\|q\|} \right\|_2$
  - Both translation loss and rotation loss were used at the same time
  - Each part of the regression helps one another
  - β is a scale factor chosen to keep the expected value of position and orientation errors to be approximately equal.
  - β was chosen depending on the scene.

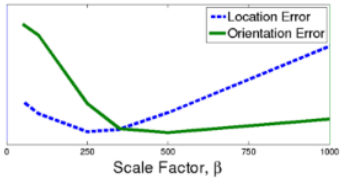


Figure 2: Relative performance of position and orientation regression on **a single convnet with a range of scale factors** for an indoor scene, Chess. This demonstrates that learning with the optimum scale factor leads to the convnet uncovering a more accurate pose function.

[1] Kendall et al., "PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalization", ICCV15
[2] Brachmann et al., "DSAC - Differentiable RANSAC for Camera Localization", CVPR17
[3] Brahmbhatt et al., "Geometry-Aware Learning of Maps for Camera Localization", CVPR18
[4] Sola, "Quaternion kinematics for the error-state Kalman filter", 2017

I – PoseNet: The loss

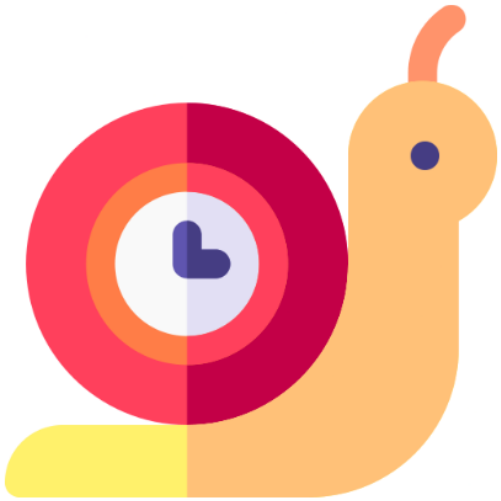
```
# Define the loss function
class BalancedMSELoss(nn.Module):
    def __init__(self):
        super(BalancedMSELoss, self).__init__()
        self.mse = nn.MSELoss()

    def forward(self, pred, target):
        trans_pred, rot_pred = pred[:, :3], pred[:, 3:]
        trans_target, rot_target = target[:, :3], target[:, 3:]
        trans_loss = self.mse(trans_pred, trans_target)
        rot_loss = self.mse(rot_pred, rot_target)
        loss = trans_loss + 500*rot_loss
        return loss
```

Question:

Instead of setting the balancing term manually, can we predict it automatically?

I – PoseNet: Training

The entire training code is located in 1-posenet_training.ipynb



You do not need to retrain the network as it might be too slow

- Metrics -

Rotational and translational errors

II – PoseNet: Metric

The training demonstrates an average MSE of ~400, Yeah!!

Wait a minute? What does it even mean?

This metric is meaningless, we would like to know the rotational and translational error for each image

Median localization results on test set

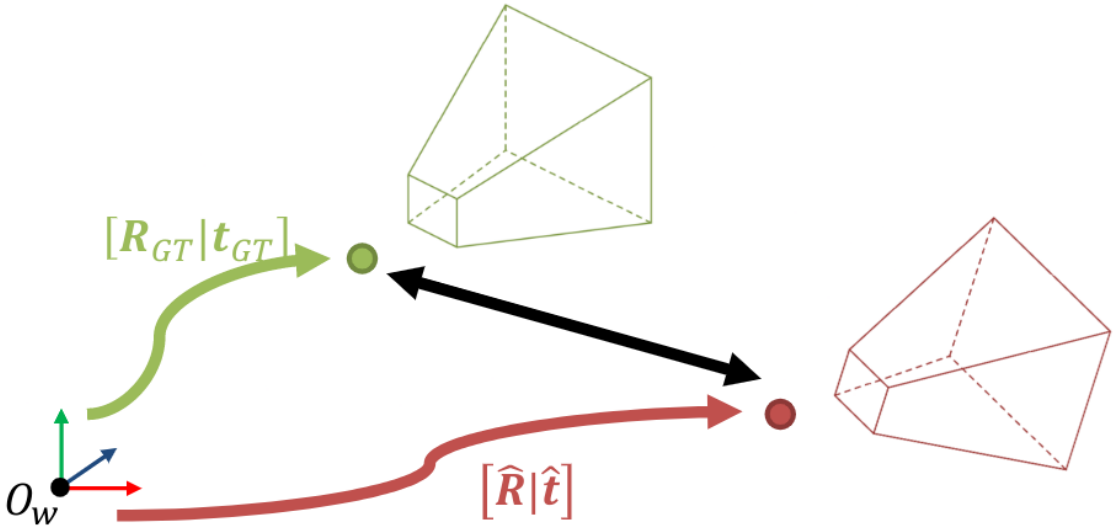
Scene	# Frames Train	# Frames Test	Spatial Extent (m)	SCoRe Forest (Uses RGB-D)	Dist. to Conv. Nearest Neighbour	PoseNet	Dense PoseNet
King's College	1220	343	140 x 40m	N/A	3.34m, 2.96°	1.92m, 2.70°	1.66m, 2.43°
Street	3015	2923	500 x 100m	N/A	1.95m, 4.51°	3.67m, 3.25°	2.96m, 3.00°
Old Hospital	895	182	50 x 40m	N/A	5.38m, 4.51°	2.31m, 2.69°	2.62m, 2.45°
Shop Façade	231	103	35 x 25m	N/A	2.10m, 5.20°	1.46m, 4.04°	1.41m, 3.59°
St Mary's Church	1487	530	80 x 60m	N/A	4.48m, 5.65°	2.65m, 4.24°	2.45m, 3.98°

II – PoseNet: Metric

You will implement these metrics in 2-posenet_metric.ipynb given these formulas

$$d_{trans}d(\boldsymbol{t}_{GT}, \hat{\boldsymbol{t}}) = \|\boldsymbol{t}_{GT} - \hat{\boldsymbol{t}}\|_2 = \sqrt{(\boldsymbol{t}_{GT}^x - \hat{\boldsymbol{t}}^x)^2 + (\boldsymbol{t}_{GT}^y - \hat{\boldsymbol{t}}^y) + (\boldsymbol{t}_{GT}^z - \hat{\boldsymbol{t}}^z)}$$

$$d_{rot}(\boldsymbol{t}_{GT}, \hat{\boldsymbol{t}}) = \text{acos}(0.5(tr(\hat{\boldsymbol{R}}^{-1}\boldsymbol{R}_{GT}) - 1))$$



II – PoseNet: Metric

Tips

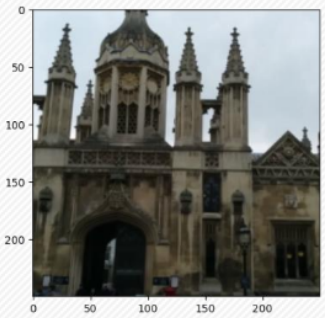
1. The conversion code from quaternion to rotation matrix is provided

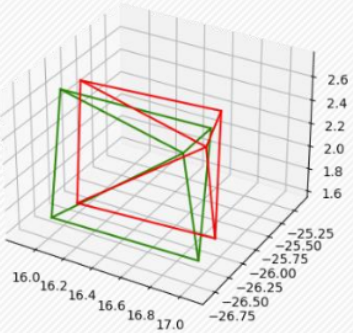
```
R1 = quaternion_to_rotation_matrix(q1)
```

2. You can select different images of the test set by adjusting this variables

```
image_number = 54  # Define the image number you want to test
```

3. The code to display the image and the 3D frustum (GT vs prediction) are provided





Try with different images

- Outliers -

Study the corner cases

III – PoseNet: Outliers

Now that you have the ability to quantify the rotational and translational error: 3-posenet_outliers.ipynb

Task 1: store all the rotational and translation errors in lists for the entire testing set

Task 2: display the median error for the translation and rotation, is it different from the original paper? Why could be the cause?

Task 3: display the images of the worst 20 estimations, what do you see?

Task 4 (Optional): Try to compute the Hard/Medium/Easy percentage of success with PoseNet? What can we deduce?

(0.25m, 2°) / (0.5m, 5°) / (5m, 10°)

High medium low - precision

- Robustness -

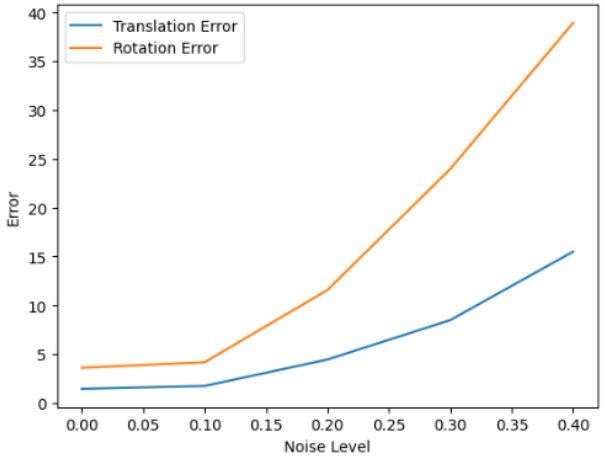
Study the effect of noise

IV – PoseNet: Robustness

We can now challenge PoseNet with a bit of image noise
We will add random noise on every test images with different standard deviations

```
noise_std_level = [0., 0.1, 0.2, 0.3, 0.4]
```

You do not have to implement anything as the run can take some time. For reference, here is the curve you should be able to see




- Transferability -
Test on unseen data

V – PoseNet: Test on unseen images

Estimate Camera Pose with Different Dataset: Cross Domain Evaluation

- Dataset : **The Visual Localization Benchmark [1]**
- Select two images with appearance changes.
- Estimate camera pose with 5-posenet_unseen_data.ipynb
- Discuss if the estimated pose is reliable or not*.
- Discuss if estimated poses are similar or not.

Illuminance Change

Seasonal Change

[1] Sattler, Benchmarking 6DOF Outdoor Visual Localization in Changing Conditions, CVPR 2018

* Posenet is currently pre-trained with Cambridge Landmarks dataset, the dataset[1] is totally unseen data. Is it possible to estimate the absolute camera pose of unseen environment?