

Practices in Visual Computing II (Spring 2025)

Assignment 1

Pose Regression with PoseNet

Total points: 50 + 2.5 points

Due: Friday, 24th January, 23:59 pm

Overview

In this assignment, you will be implementing PoseNet, a type of pose regression network. Your network is trained on a dataset of images and the positions and orientations of the camera during capturing those images, and then it needs to predict position/orientations for new unseen images (the new images should be from the same data distribution of the training images). It has applications in AR/VR, graphics, and autonomous systems industries.

You can learn about the task by watching the **PoseNet video** or study the **related paper**.

1 Introduction (0 Points)

At its core, PoseNet is a convolutional neural network and uses convolutional layers to learn the features of the training images. These features are then used to learn specific properties present in the images, such as the camera position. CNNs tend to be very deep and hard to train, especially without large GPUs. Because of this, PoseNet relies on a pre-trained "general-purpose feature extractor". For this, it uses a modified version of InceptionV1 that was pretrained on the Places dataset. The original architecture of InceptionV1 can be seen in Figure 3.

2 Workspace Initialization (0 Points)

Download the KingsCollege dataset from **here** and extract it in the `data/datasets/` directory.

3 Inception Block (10 Points)

Implement the architecture of the InceptionV1 backbone in `PoseNet.py`. Figure 1 shows all the parameters needed to initialize the main structure of Inception, without the loss paths. All convolutions and max pooling layers (not the average pooling layers) use padding $p = \frac{k-1}{2}$. Also make sure to use ReLU after every convolution, max pooling, and average pooling layer.

(Hint: Use `nn.Sequential` blocks to simplify the model definition. In a sequential block, `nn.Flatten()` can easily convert the output of a convolutional layer into the input of a linear layer.)


type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Figure 1: InceptionV1

4 PoseNet Architecture (10 Points)

The loss paths are slightly modified in PoseNet from the Inception original (See Figure 4 for details):

1. Remove all three softmax layers with their preceding linear classifier layers. Replace them with 2 parallel linear layers. One with an output size of 3 to predict the xyz position, and another one with an output size of 4 to predict the $wpqr$ orientation. The position is predicted as a 3D coordinate and the orientation as a Quaternion in $wpqr$ ordering.
2. In the final (third) loss header, insert another linear layer with output dimension 2048. This output will then be used as the input for the two linear layers described in the previous point.



type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj
convolution	7×7/2	112×112×64	1						
max pool	3×3/2	56×56×64	0						
convolution	3×3/1	56×56×192	2		64	192			
max pool	3×3/2	28×28×192	0						
inception (3a)		28×28×256	2	64	96	128	16	32	32
inception (3b)		28×28×480	2	128	128	192	32	96	64
max pool	3×3/2	14×14×480	0						
inception (4a)		14×14×512	2	192	96	208	16	48	64
inception (4b)		14×14×512	2	160	112	224	24	64	64
inception (4c)		14×14×512	2	128	128	256	24	64	64
inception (4d)		14×14×528	2	112	144	288	32	64	64
inception (4e)		14×14×832	2	256	160	320	32	128	128
max pool	3×3/2	7×7×832	0						
inception (5a)		7×7×832	2	256	160	320	32	128	128
inception (5b)		7×7×1024	2	384	192	384	48	128	128

padding = (kernel_size - 1) / 2

Figure 2: Inception layer parameters

Training the Inception network takes a long time. Fortunately, it has been done before, so we can load the weights for the layers we are reusing and use them to initialize the model. In the initialization of PoseNet, the framework already loads the weights from a file. They are stored as a dictionary, so you can check the available layers using `print(weights.keys)`. Use the function `init(key, module, weights)` to help you initialize a new layer with the pre-trained weights.

(Hint: You will need to use all the loaded weights from the pre-trained Inception, except for the classifier layers.)

Complete `LossHeader` and `PoseNet` classes in the `PoseNet.py`.

5 Loss function (10 Points)

The PoseNet architecture has three loss headers. Each of these loss headers predicts an xyz position and a $wpqr$ orientation. The position is predicted as a 3D coordinate and the orientation as a Quaternion in $wpqr$ ordering. We will calculate a loss for each loss header individually and then add them together to build the final loss. The loss is given by

$$\text{loss}_i = \|\mathbf{x}_i - \mathbf{x}_{gt}\|_2 + \beta \left\| \mathbf{q}_i - \frac{\mathbf{q}_{gt}}{\|\mathbf{q}_{gt}\|} \right\|_2 \quad (1)$$

$$\text{loss} = w_1 \times \text{loss}_1 + w_2 \times \text{loss}_2 + w_3 \times \text{loss}_3. \quad (2)$$

The parameters β define if the model focuses more on reducing the position or the orientation error, while w_i controls how much influence the auxiliary losses have. We will use these values for the parameters:

$$\beta = 300 \quad (3)$$

$$\mathbf{w} = (0.3, 0.3, 1) \quad (4)$$

Implement the loss in the class `PoseLoss` in `PoseNet.py`.

6 Dataloader (10 Points)

To train the network we will use the Kings College dataset. The `DataSouce.py` already loads the dataset and provides the structure for serving the batches. Your task is to implement the image preprocessing pipeline in the `DataSource.py`. (*Hint: Use `torchvision.transforms`.*)

1. **Resize:** Resize the images to make the smaller dimension equal to 256 (e.g. 1920x1080 to 455x256).
2. **Subtract a mean image:** To boost the training, a mean image I_{mean} needs to be subtracted from each image. I_{mean} needs to be precomputed. For this, finish `generate_mean_image()` as follows:
 - Load each image.
 - Resize as in step 1.
 - Add them together.
 - Normalize values by the number of images.

The precomputed mean image I_{mean} needs to be subtracted from each image when serving the images to the training/testing loop.

(*Hint: Subtracting images can easiest be done by converting them to numpy arrays. Note that numpy stores images as (height, width), while PIL stores images as (width, height).*)

3. **Crop:** InceptionV1 is designed for a 224x224 input, so we need to crop the images to this size.
 - During training: Crop a random 224x224 piece out of the image.
 - During testing: Crop the 224x224 center out of the image.
4. **Normalize:** Use a mean of 0.5 and standard deviation of 0.5 to normalize each channel of the images.

7 (Optional) Netron (2.5 Points)

For this part, you will be using the **Netron** tool for visualizing your architecture. Export a PNG of your implemented architecture with your student ID as the network's input name.

8 Demo (10 Points)

During the demo session, first, we will check if you implemented all of the other parts. Then, we will ask you exactly four questions related to the assignment, each having 2.5 points.

- (a) Your code needs to be written in a clear, modular format. We ask you to run your code. Your training/inference code needs to run without errors
- (b) Your code needs to produce reasonable results:
 - i. Median positional error within 5 meters
 - ii. Median orientation error within 5 degrees
- (c) Then we will ask you a question about a part of the code. You need to be able to clearly explain what's going on
- (d) Finally, we will ask a (fairly simple) question to test your knowledge and intuition about the assignment

Note that your codes should be zipped and submitted to Canvas before the deadline and you may not change them during the demo session day.

