Simon Fraser University

Camera Localization

Johannes Merz, Ali Mahdavi Amiri, Mehdi Safaee

February 20, 2022

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 being 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 pretrained "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 6.1.

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

1 INITIALIZE YOUR WORKSPACE (0 POINTS)

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

2 INCEPTION BLOCK (4 POINTS)

Implement the architecture of the InceptionV1 backbone in **PoseNet.py**. Figure 3.1 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.)*

3 POSENET ARCHITECTURE (6 POINTS)

The loss paths are slighty modified in PoseNet from the Inception original (See Figure 6.2 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.

kernel_size = 1

padding = (kernel_size – 1) / 2

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

Figure 3.1: 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**.

4 LOSS FUNCTION (5 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

$$\text{loss}_i = \left\|\mathbf{x}_i - \mathbf{x}_{gt}\right\|_2 + \beta \left\|\mathbf{q}_i - \frac{\mathbf{q}_{gt}}{\|\mathbf{q}_{gt}\|}\right\|_2 \tag{4.1}$$

$$\text{loss} = w_1 \times \text{loss}_1 + w_2 \times \text{loss}_2 + w_3 \times \text{loss}_3. \tag{4.2}$$

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 auxilliary losses have. We will use these values for the parameters:

$$\beta = 300 \tag{4.3}$$

$$\mathbf{w} = (0.3, 0.3, 1) \tag{4.4}$$

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

5 DATALOADER (5 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.

6 Experiments and Report (5 Points)

Define proper hyperparameter settings, train and experiment with the model in **train.py and test.py** and report your results.

Submission checklist:

- **report.pdf**
- **a .zip containing all *.py files**

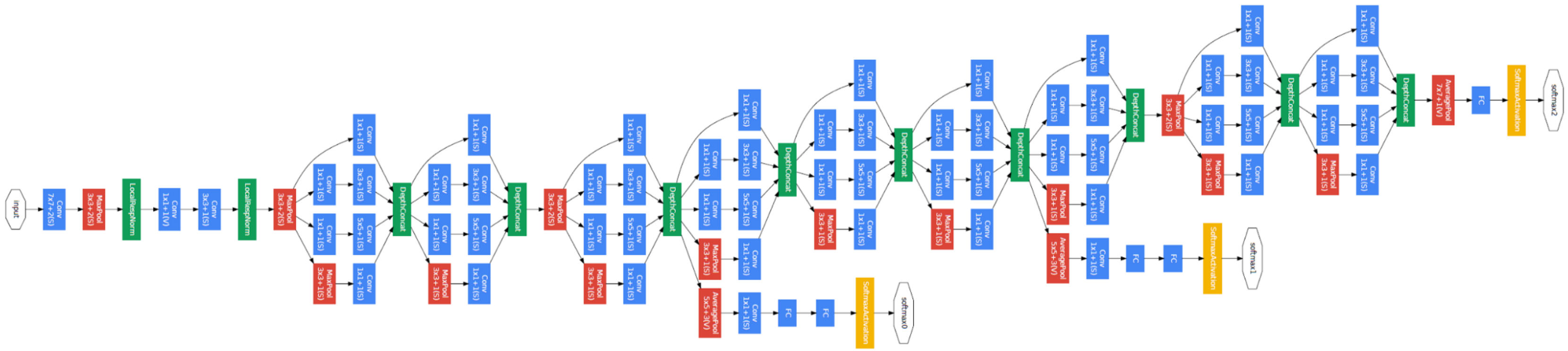

Figure 6.1: InceptionV1

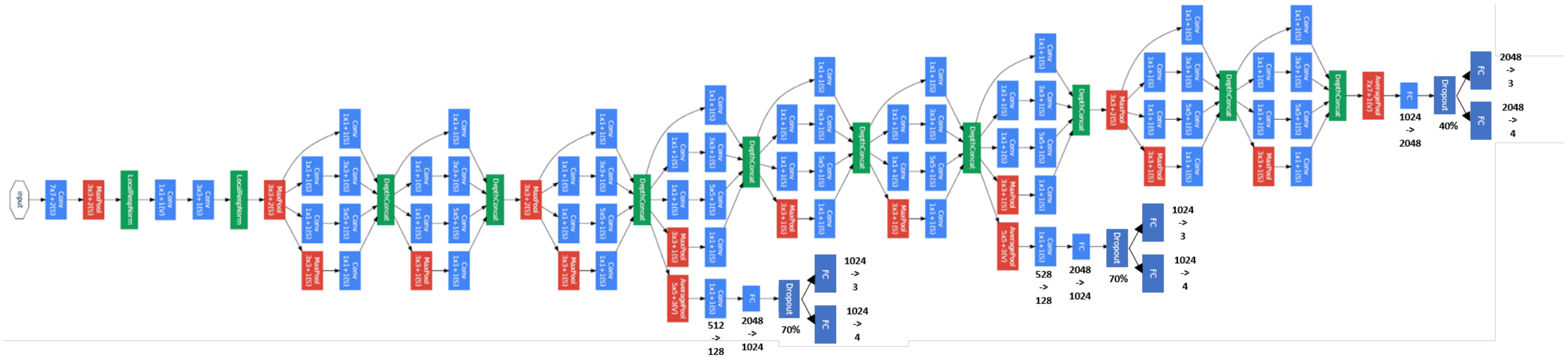

Figure 6.2: PoseNet