INFOMCV 2021 practice exam solutions

1. **Augmented reality app**

This is essentially inside-out tracking. The camera intrinsic matrix is already known. If we know the size of a tile of the chessboard (required part 1/2), we can recover the camera's position relative to the chessboard's upper-left corner using solvePnP (required part 2/2).

2. **Silhouette-based volume reconstruction from templates**

a. The templates are 80 pixels high but people can be between 80-160 pixels high. So, we will have to do some scaling. The most straightforward approach is to do template matching. For each "hit", you "paste" the mask in a foreground image for that view. Once you're done with all views, the processing steps are what you've implemented for Assignment 2.

We need a (maximum) threshold for the distance between two HOG descriptors (thresh_HOG). **I**nitialize all foreground images to be 640x480 single-channel and all zeros.

Then loop over all scales (e.g., with scaling factor 1 to 2 with increments of 10%), starting columns and starting rows. Calculate the HOG descriptor from image patch (HOG_temp).

Then loop over all templates. Compare HOG_temp to the current T_HOG. If the distance is below thresh_HOG, pad T_mask to 640x480 according to starting position and scale (pad_mask). Then perform an OR operation of the current T_mask and pad_mask.

The remainder of the pipeline is as in Assignment 2. That is, loop over all voxels. For each voxel, project to all views. When the projected pixels is "on" in all views, set the voxel to "on".

b. If there can be false positives, it wouldn't be wise to just add the mask to the foreground image. In this case, store the match (view, location, scale, template, HOG distance). After template matching, apply the non-maximum suppression algorithm (e.g., with IoU of 0.5). Only keep those detections that are above a (conservative) threshold. Loop over the remaining detections, pad the corresponding template and OR it with the corresponding view. Then proceed with the voxel reconstruction as in (a).

3. **Chess piece recognition**

a. Mirroring is useful because SIFT descriptors are not mirror-invariant
Rotation is not useful because SIFT features are rotation invariant so the effect is cancelled
Contrast adjustment is useful because it affects the relative gradients of the SIFT descriptor

b. Initialize an empty vector/array/list to store SIFT descripts (SIFT_list)

Loop over all 250 images and all tiles. If the label of the tile corresponds to an actual piece, calculate SIFT descriptors and store them in SIFT_list.

Then, cluster all SIFT descriptors using k-means with 100 cluster centers. The cluster centers form the codebook.

4. Fish detection

a. We start with a lot of clusters. These will be too small to have a sufficient overlap with the annotated fish. So recall is 0. Precision is undefined (so zero). When we merge, the regions become larger. This means that, up to some point, it is more likely that we have an actual match. So recall increases. Precision also increases because we have more true positives and fewer false negatives. But then both recall and precision drop because the regions get so large that the IoU is always less than 0.5.

b. With a low IoU, it's easy to match a fish, hence high recall. Moreover, even detections that only very slightly overlap with the ground truth was be counted as correct. When IoU increases, the number of true positives decreases. This means that recall decreases. Also, there will be more false positives because detections will more often not match the ground truth, so precision decreases.

5. Gesture recognition two-stream CNN

a. The trick is to have the same spatial extent for the two streams at the mid-level fusion layer. One option is to go for 32x32. This means that the RGB stream need to go from 64 to 32. You can do this by having a first and second conv layer with 3x3 kernels and padding=1, stride=1. This means there is no change in the spatial extent. Then the pooling layer can be max-pool with size 2x2 and stride 2 and no padding. This will reduce the spatial extent by 2 in both directions.

For the OF stream, we need to go from 40 to 32. We can do a first 5x5 conv with padding = 0 and stride = 1. This will result in an output activation size of 36x36. We then perform a max-pool operation with 3x3 kernel and stride = 1. With padding = 0, this will result in an output of 34x34. Finally, we do a convolution with 3x3 filters, padding = 0 and stride = 1 to arrive at a 32x32 output.

The only other hyper-parameter that we need to determine is the type of global pooling, which can be average pooling or max pooling.

Many other options are possible.

b. To continue with the example in (a):

Layer	Parameters	Size
RGB: input	-	64x64x3
RGB: conv1	(3 x 3 x 3 + 1) x 16	64x64x16
RGB: pool1	-	32x32x16
RGB: conv2	(3 x 3 x 16 + 1) x 16	32x32x16
OF: input	-	40x40x16
OF: conv1	(5 x 5 x 16 + 1) x 16	36x36x16
OF: pool1	-	34x34x16
OF: conv2	(3 x 3 x 16 + 1) x 16	32x32x16
Fusion	-	32x32x32
Global pooling	-	32x32x1
Fully connected	(32 x 32 + 1) x 100	100x1
Output	(100 + 1) x 5	5x1

6. Instagram CycleGAN and transfer learning

a. For CycleGAN, you need to have to sets of (unpaired) inputs. You already have a set of Instagram-filtered images. Now we need a set of "normal" images, taken with your phone. So shoot a number of photos like you would normally do. Then train the CycleGAN using the set of Instagram-filtered photos as domain 1 and the "normal" photos as domain 2 (or the other way around). Once the training has completed, use the mapping from domain 1 to domain 2 with the Instagram-filtered photos and the output should be "normal" looking photos of the same scenes.

b. The training data is generated by cropping the Instagram photos using the labeled bounding boxes. Resize the crop to 224x224 and assign the manually annotated labels to it. Your CNN is identical, so the weights are your initialization. You train it using the training data as you would normally train a CNN.