Exam in SSY097

August 19th, 2019

Allowed materials : Pen/pencil, eraser.

The exam consists of six problems. Make sure that you have them all.

- Motivate all answers carefully.
- Use a new paper for each new numbered problem.
- Write on one side of the papers only.
- Write your anonymous number on each new page.
- Avoid using a red pen.
- If you want the result registered as SSY096, write this on the cover page.

The dates for the exam review will be announced on PingPong.

Grades

$\geq$ **8 points** Grade: 3

$\geq$ **11 points** Grade: 4

$\geq$ **14 points** Grade: 5

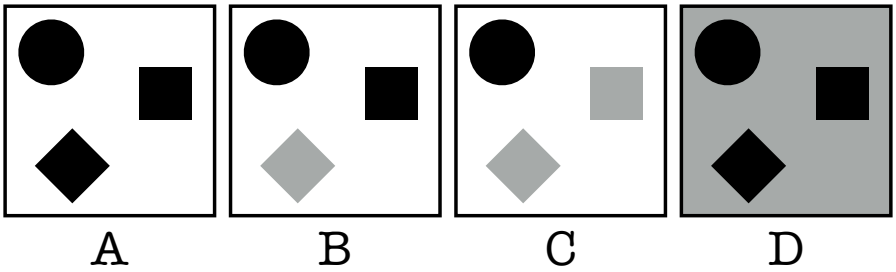
1 SIFT, 3 points

(a) Given a patch centered around a keypoint, SIFT computes a descriptor from that patch that can be used for image matching.
Describe how SIFT computes these descriptors: Name the key steps in the descriptor computation process and describe each in 1-2 sentences.

(Solution) SIFT uses the following steps to extract a descriptor from the patch:

- Gradient computation: For each pixel in the patch, the gradient direction and magnitude is computed.
- Subdivision: The patch is subdivided into 4×4 regions of equal size.
- Histogram computation: For each region, a histogram over the gradient directions with 8 bins is computed, where each pixel in the region contributes the magnitude of its gradient to its corresponding bin.
- Descriptor computation: All 4×4 8-dimensional histograms are concatenated to form a 128-dimensional descriptor.
- Normalization: The resulting descriptor is normalized such that is has unit L2 norm.

(b) Which of the following patches will result in the same SIFT descriptor? Justify your answer.



(Solution) Due to the L2 normalization step, two patches will result in the same SIFT descriptor if the gradient direction for each pixel is the same and the gradient magnitudes are related by a constant scalar factor. This is the case for patches $\mathcal{A}$ and $\mathcal{D}$, which will thus result in the same descriptor. In contrast, patches $\mathcal{B}$ and $\mathcal{C}$ cannot be related to any other patch by a single scaling factor, thus resulting in different descriptors.

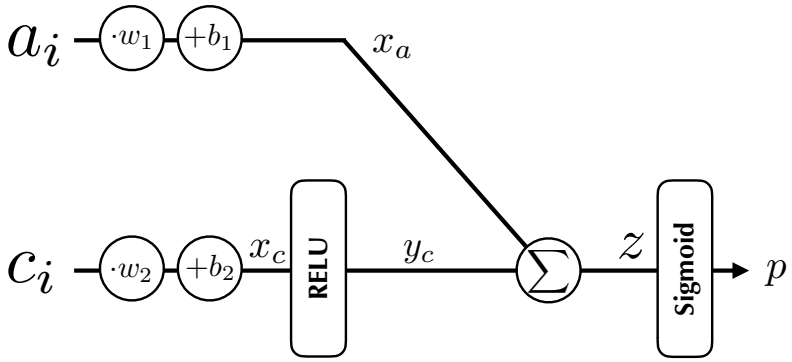
(c) Prior to descriptor computation, SIFT aligns the patch from which the descriptor is extracted to the orientation assigned to the keypoint.
Describe in 2-3 sentences how the orientation of the keypoint is determined.

(Solution) Given a region around the keypoint, gradients are computed for each pixel in that region. SIFT then generates a histogram over gradient orientations, where each pixel contributes with the magnitude of its gradient to its corresponding histogram bin. SIFT uses the orientation corresponding to the bin with maximal weight as the orientation of the keypoint (a better orientation can be obtained by linear interpolation with the neighboring bin).

2 Statistical Learning, 3 points

Consider the neural network shown below that performs binary classification on a tuple (a_i, c_i) of two scalar input values $a_i, c_i \in \mathbb{R}$. The RELU and Sigmoid functions are given as $\text{RELU}(x) = \max(0, x)$ and $\text{Sigmoid}(z) = \frac{e^z}{1+e^z}$. The derivative of the RELU function is given as

$$\frac{\partial \text{RELU}(x)}{\partial x} = \begin{cases} 1 & \text{if } x > 0, \\ 0 & \text{otherwise} \end{cases} .$$



(a) Provide the loss functions L_i for when the example (a_i, c_i) is positive and when it is negative. The loss should depend on the probability p estimated by the Sigmoid layer. As in the lecture and the lab, use the negative log-likelihood loss.

(Solution) Let L_i be the loss for the example. For a positive example, we want to maximize p, i.e., the probability computed by the classifier that the example is a positive one. The negative log-likelihood loss is thus given by $L_i = -\ln(p)$.
For a negative example, we want to maximize $1-p$. The negative log-likelihood loss is thus given by $L_i = -\ln(1-p)$.

(b) Given a positive example $(a_i, c_i) = (1, 2)$, compute the derivatives

$$\frac{\partial L_i}{\partial w_1}, \qquad \frac{\partial L_i}{\partial b_1}, \qquad \frac{\partial L_i}{\partial w_2}, \qquad \frac{\partial L_i}{\partial b_2}$$

through the backpropagation algorithm. To this end, first perform a forward pass to compute values for x_a, x_c, y_c, z, and p. Next, use the chain rule to derive formulas for the derivatives in the backward pass. Compute the actual values for the derivatives using your equations and the values for x_a, x_c, y_c, z, and p computed during the forward pass.
The current values for w_1, b_1, w_2, and b_2 are

$$w_1 = -15, \qquad b_1 = 5, \qquad w_2 = 4, \qquad b_2 = 2 \ .$$

(Solution) We have the following relations:

$$\begin{aligned} x_a &= w_1 \cdot a_i + b_1 \\ x_c &= w_1 \cdot c_i + b_1 \\ y_c &= \max(0, x_c) \\ z &= x_a + y_c \\ p &= \frac{e^z}{1+e^z} \end{aligned}$$

The forward pass yields the following values:

$$\begin{aligned} x_a &= -10 \\ x_c &= 10 \\ y_c &= 10 \\ z &= 0 \\ p &= 0.5 \end{aligned}$$

We need the following derivatives for the Sigmoid and the RELU:

$$\begin{aligned} \frac{\partial p}{\partial z} &= \frac{e^z(1+e^z) - (e^z)^2}{(1+e^z)^2} \\ &= \frac{e^z}{1+e^z} \cdot \frac{1}{1+e^z} = p \cdot (1-p) \\ \frac{\partial y}{\partial x} &= \begin{cases} 1 & \text{if } x > 0, \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

The chain rule provides the following relationships for the gradients:

$$\begin{aligned} \frac{\partial L_i}{\partial z} &= \frac{\partial L_i}{\partial p}\frac{\partial p}{\partial z} = -\frac{1}{p} \cdot p \cdot (1-p) = (p-1) \\ \frac{\partial L_i}{\partial y_c} &= \frac{\partial L_i}{\partial z}\frac{\partial z}{\partial y_c} = (p-1) \cdot 1 \\ \frac{\partial L_i}{\partial x_a} &= \frac{\partial L_i}{\partial z}\frac{\partial z}{\partial x_a} = (p-1) \cdot 1 \\ \frac{\partial L_i}{\partial x_c} &= \frac{\partial L_i}{\partial y_c}\frac{\partial y_c}{\partial x_c} = (p-1) \cdot 1 \quad \text{since } x_c > 0 \text{ in our example} \\ \frac{\partial L_i}{\partial b_1} &= \frac{\partial L_i}{\partial x_a}\frac{\partial x_a}{\partial b_1} = (p-1) \cdot 1 \\ \frac{\partial L_i}{\partial b_2} &= \frac{\partial L_i}{\partial x_c}\frac{\partial x_c}{\partial b_2} = (p-1) \cdot 1 \\ \frac{\partial L_i}{\partial w_1} &= \frac{\partial L_i}{\partial x_a}\frac{\partial x_a}{\partial w_1} = (p-1) \cdot a_i \\ \frac{\partial L_i}{\partial w_2} &= \frac{\partial L_i}{\partial x_c}\frac{\partial x_c}{\partial w_2} = (p-1) \cdot c_i \end{aligned}$$

Using the values computed during the forward pass then yields

$$\begin{aligned}
\frac{\partial L_i}{\partial b_1} &= -0.5 \\
\frac{\partial L_i}{\partial w_1} &= -0.5 \\
\frac{\partial L_i}{\partial b_2} &= -0.5 \\
\frac{\partial L_i}{\partial w_2} &= -1
\end{aligned}$$

(c) Using a learning rate of 0.1, what is the next value for w_2?

(Solution) Using a learning rate of 0.1, the next value for w_2 is given as

$$w_2 = w_2 - 0.1 \cdot \frac{\partial L_i}{\partial w_2} = 4 + 0.1 = 4.1$$

3 (Robust) Model Fitting, 3 points

(a) A 2D affine transformation maps a 2D point $p_1 = (x_1, y_1)$ to a 2D point $p_2 = (x_2, y_2)$ via

$$\begin{pmatrix} x_2 \\ y_2 \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix} \quad . \tag{1}$$

Explain how to construct a solver that can be used inside RANSAC to estimate a transformation between two sets of 2D points. To this end, derive a linear system $\mathtt{M}\theta = \mathbf{v}$, where the vector θ contains the parameters of the transformation.
It is sufficient to derive the equations for a single correspondences between p_1 and p_2.

How many correspondences are needed for your minimal solver?

(Solution) A correspondences between p_1 and p_2 provides two linear equations in the six unknowns a, b, c, d, t_x, and t_y:

$$\begin{pmatrix} x_1 & y_1 & 0 & 0 & 1 & 0 \\ 0 & 0 & x_1 & y_1 & 0 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \\ t_x \\ t_y \end{pmatrix} = \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} \quad .$$

Stacking the equations for multiple correspondences then yields the linear system $\mathtt{M}\theta = \mathbf{v}$, where $\theta = (a, b, c, d, t_x, t_y)^T$.

Since there are two equations in six unknowns per correspondence, three correspondences are needed by the minimal solver.

(b) How does Eq. 1 change when estimating a homography rather than an affine transformation?
Make sure to explain all variables.

(Solution) The equation for a homography is

$$\lambda \begin{pmatrix} x_2 \\ y_2 \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix} \quad , \tag{2}$$

where $\lambda \neq 0$ is a scaling factor since a homography is a perspective transformation and operates on homogeneous coordinates.

(c) Methods for least-squares model fitting implicitly assume that the noise on the data points follows a Gaussian / Normal distribution. Thus, they are not able to handle outlier measurements as these do not follow a Gaussian / Normal distribution.
Explain in 2-3 sentences how robust cost functions address this problem.

(Solution) The problem with a Gaussian / Normal distribution is that outliers are very unlikely, i.e., these data points result in a high cost during optimization. Robust cost functions thus use probability distributions for which outliers are more likely. As a result, the cost of outliers during optimization is reduced, thus reducing their influence on the result.

4 RANSAC, 3 points

In the following, we will consider the problem of fitting a 2D line through a set of 2D points in the presence of outliers. We will use a minimal solver that fits a line through two points.

(a) Given an inlier ratio of ε, derive a formula for the number of RANSAC iterations k required such that the probability of missing the best model is at most 1%.
Explain how you obtain your formula.

(Solution) The probability of selecting a sample that does not contain two inliers in a single RANSAC iteration is $(1 - \varepsilon^2)$. Consequently, the probability of not selecting a sample that contains two inliers in each of k iterations is $(1 - \varepsilon^2)^k$. We want that $(1 - \varepsilon^2)^k \leq 0.01$, resulting in the following equation:

$$k \geq \frac{\ln 0.01}{\ln(1 - \varepsilon^2)} \quad . \tag{3}$$

(b) Implement RANSAC for 2D line fitting in pseudo code or MATLAB code (or a mixture of the two). You can assume that you have a function `line = minimal_solver(S)` that computes a line hypothesis from a minimal sample `S` and a function `squared_error = evaluate(line, c)` that returns the squared error when evaluating a line hypotheses `line` on a 2D point correspondences `c`.
Make sure that you explain all variables that you use.

(Solution)

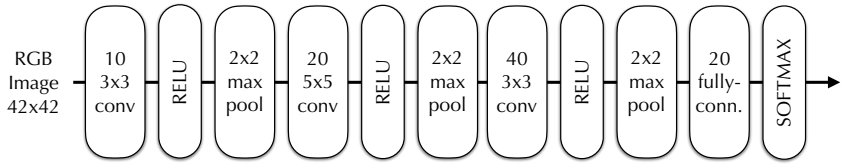
```
line_best = RANSAC(C, tau)
    ε = 2/n    // initialize inlier ratio
    max_inliers = 0    // initialize max. number of inliers
    k_max = log(0.01)/log(1−ε²)    // initialize max. number of iteration
    for t = 1 : k_max
        S = select 2 matches at random from C
        line = minimal_solver(S)
        inliers = 0
        for i = 1 : n
            error = evaluate(line, C(i))
            if error < tau
                inliers = inliers + 1
        if inliers > max_inliers
            max_inliers = inliers
            ε = max_inliers/n
            k_max = log(0.01)/log(1−ε²)
            line_best = line
    Return line_best
```

Here, `C` is a set of n 2D-2D correspondences and `C(i)` denotes the i-th correspondence. `tau` is a threshold on the squared error between a point and a line that is used to distinguish between inliers and outliers.

(c) In a general context (not necessarily for line fitting), assume that a minimal solver requires only a single data point as input.
How many RANSAC iterations are necessary to ensure that RANSAC finds the best solution with probability 100%? Justify your answer!

(Solution) Assume we are given n data points. Since a single model hypothesis can be computed from a single data point, there are at most n different model hypotheses. In order to ensure that RANSAC finds the best model, it is thus sufficient to create a model hypotheses for each data point and then evaluate it on all data points. In other words, we replace RANSAC's random sampling step by a deterministic sampling that returns one point after the other.

5 Deep Learning, 3 points



(a) Consider the convolutional neural network shown above. For each layer of the network with trainable parameters, write down the number of trainable parameters.
Assume that the convolutional layers do not use padding and that no bias terms are used.

(Solution) The RELU and max. pooling layers, as well as the SOFTMAX layer do not have trainable parameters.

The number of trainable parameters of m $k \times k$ convolutions applied on a volume of size $W \times H \times D$ is $m \cdot k \cdot k \cdot D$. Thus, we have the following number of trainable parameters for the fully convolutional layers: 270 parameters for the 10 3×3 conv. layer, 5,000 parameters for the 20 5×5 conv. layer, and 7,200 parameters for the 40 3×3 conv. layer.

The number of trainable parameters in a fully connected layers depends on the size $W \times H \times D$ input volume and the output dimensionality D'. In the example above, the intermediate volumes are:
$42 \times 42 \times 3$ (input), $40 \times 40 \times 10$ (after first conv.), $40 \times 40 \times 3$ (after RELU), $20 \times 20 \times 10$ (after max. pooling), $16 \times 16 \times 20$ (after second conv. and RELU), $8 \times 8 \times 20$ (after max. pooling), $6 \times 6 \times 40$ (after third conv. and RELU), $3 \times 3 \times 40$ (after max. pooling)
The fully connected layer thus has $3 \cdot 3 \cdot 40 \cdot 20 = 7{,}200$ parameters.

(b) Assume that you replace the 20 5×5 conv. block in the network above by a 20 7×7 conv. block. What is the number of trainable parameters in the fully connected layer?

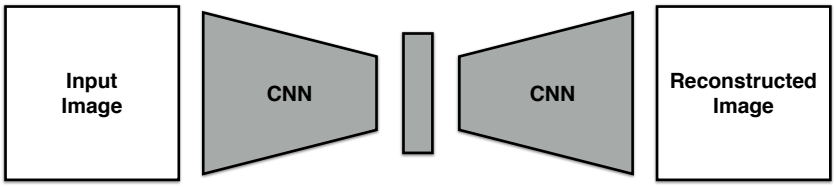
(Solution) In the case of the 7×7 conv. block, the dimensions of the intermediate volumes are:
$14 \times 14 \times 20$ (after 7×7 conv. and RELU), $7 \times 7 \times 20$ (after max. pooling), $5 \times 5 \times 40$ (after third conv. and RELU), $2 \times 2 \times 40$ (after max. pooling)
The fully connected layer thus has $2 \cdot 2 \cdot 40 \cdot 20 = 1{,}600$ parameters.

(c) Assume that you replace the fully connected layer in the network from a) by a fully convolutional layer to be able to apply the network on RGB input images of arbitrary dimensions. You do the replacement such that applying the new network without padding on a 42×42 RGB image results in the same output as applying the original network.
How many trainable parameters does the new fully convolutional layer have? Explain your answer!

(Solution) In order to obtain the same result, the fully convolutional layer performs the same operation as the fully connected layer on a $3 \times 3 \times 40$ input volume. Thus, the layer has $3 \cdot 3 \cdot 40 \cdot 20$ = 7,200 parameters.

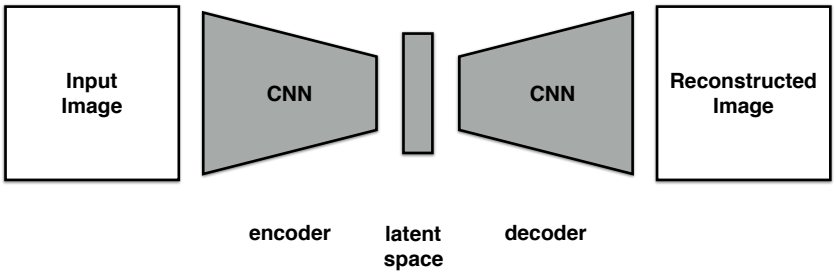
6 Generative Networks, 3 points

(a) The general structure of a Variational Autoencoder is given in the figure below.



Name the different parts (marked in gray) and explain their purpose (a single sentence per component is sufficient).
Which parts are required during training? Which parts are used during testing?

(Solution) The names of the individual parts are given below.



The encoder maps an input image to a point in the latent space. The latent space is trained to represent the space of all natural images. In turn the decoder maps points in the latent space to images.
During training, all parts (encoder, latent space, decoder) are required. During testing, one randomly samples a point in the latent space and uses the decoder to reconstruct an image from it. Thus, the encoder is not required during testing.

(b) The training objective of a generative model is given by

$$\min_G \max_D E_{\mathbf{x}\sim p_{\text{data}}}[\log(D(\mathbf{x})] + E_{\mathbf{z}\sim p_{\text{model}}}[\log(1 - D(G(\mathbf{z})))] \ . \tag{4}$$

Is Eq. 4 the training objective of a VAE or a Generative Adversarial Network (GAN)?
Explain all terms in Eq. 4.

(Solution) Eq. 4 is the training objective of a GAN, where G and D correspond to the generator and discriminator of the GAN, respectively. $D(X)$ corresponds to the probability that a real image X is considered real by the discriminator. The discriminator aims to maximize this probability.
$G(Z)$ is the image reconstructed by the generator from a point Z in the latent space. Consequently, $D(G(Z))$ corresponds to the probability that a fake image $G(Z)$ is considered real by the discriminator. D tries to minimize this probability while G tries to maximize it.

(c) Assume that you are given a GAN that can generate images of dogs and cats. How can this GAN be used to help train a convolutional neural network that classifies images into cats and dogs?
Do you expect your approach to solve the problem of obtaining large training sets of labelled data in general? Explain your answer!

(Solution) The GAN can be used for data augmentation by being used to generate additional training data.
This approach is unlikely to solve the problem of obtaining labelled training data: the GAN itself was trained from limited training data. As such, it is unlikely that it is able to generate realistic images that differ too much from its training data, e.g., it is unlikely that it will generate images of types of dogs it has not seen during training.