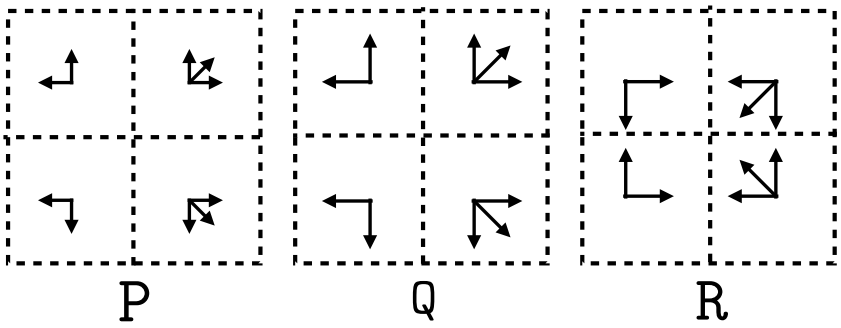
1 SIFT, 3 points

(a) For each region, we have an 8-dimensional histogram. For 4 regions, we thus obtain a 32-dimensional descriptor.
Vector bouquets:



The descriptor of `Q` differs from the descriptor of `P` by a multiplicative factor $k > 1$ caused by stronger gradients (from black to white instead of from gray to white). After L2-normalization, both descriptors are thus identical.

(b) Given a detected keypoint, SIFT estimates the dominant gradient direction in a region around the keypoint. The patch from which the SIFT descriptor is extracted is then aligned to this dominant direction before computing the descriptor. This is corresponds to a shuffling of the order of the descriptor bins. The alignment to the dominant direction makes the descriptor rotation invariant as any rotation of the image will result in the same patch.

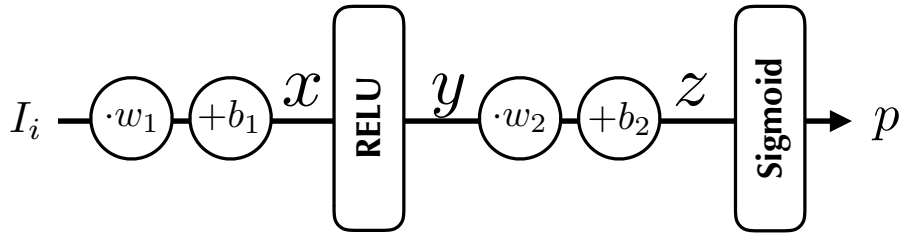
(c) The table below shows the distances of each feature in image $\mathcal{P}$ to its two nearest neighbors in image $\mathcal{Q}$:

	distance to		
	1st neighbor	2nd neighbor	ratio
p_1	2 (q_2)	4 (q_4)	1/2
p_2	1 (q_1)	2 (q_3)	1/2
p_3	0.5 (q_3)	3.5 (q_1)	1/7
p_4	1 (q_1)	3 (q_2)	1/3

The matches (p_3, q_3) and (p_4, q_1) pass the ratio test with a threshold of 0.4.

2 Statistical Learning, 3 points

Consider the simple neural network shown below that performs binary classification on scalar input values.



(a) Let L_i be the loss for a negative example I_i. We want to maximize $1 - p(I_i)$, where $p(I_i)$ is the probability computed for I_i by the classifier. The negative log-likelihood loss is thus $L_i = -\ln(1 - p(I_i))$. In the following, we will use p instead of $p(I_i)$.

(b) We have the following relationships:

$$\begin{aligned} x &= w_1 \cdot I_i + b1 \\ y &= \max(0, x) \\ z &= w_2 \cdot y + b_2 \\ p &= \frac{e^z}{1 + e^z} \end{aligned}$$

The forward pass yields the following values:

$$\begin{aligned} x &= 15 \\ y &= 15 \\ z &= 0 \\ p &= 0.5 \end{aligned}$$

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:

$$\begin{aligned}
\frac{\partial L_i}{\partial z} &= \frac{\partial L_i}{\partial p}\frac{\partial p}{\partial z} = \frac{1}{1-p}\cdot p\cdot(1-p) = p \\
\frac{\partial L_i}{\partial b_2} &= \frac{\partial L_i}{\partial z}\frac{\partial z}{\partial b_2} = p\cdot 1 \\
\frac{\partial L_i}{\partial w_2} &= \frac{\partial L_i}{\partial z}\frac{\partial z}{\partial w_2} = p\cdot y \\
\frac{\partial L_i}{\partial y} &= \frac{\partial L_i}{\partial z}\frac{\partial z}{\partial y} = p\cdot w_2 \\
\frac{\partial L_i}{\partial x} &= \frac{\partial L_i}{\partial z}\frac{\partial y}{\partial x} = p\cdot w_2 \quad \text{since } x>0 \text{ in our example} \\
\frac{\partial L_i}{\partial b_1} &= \frac{\partial L_i}{\partial x}\frac{\partial x}{\partial b_1} = p\cdot w_2\cdot 1 \\
\frac{\partial L_i}{\partial w_1} &= \frac{\partial L_i}{\partial x}\frac{\partial x}{\partial w_1} = p\cdot w_2\cdot I_i
\end{aligned}$$

Substituting the values obtained from the forward pass yields:

$$\frac{\partial L_i}{\partial w_1} = 0.5 \qquad \frac{\partial L_i}{\partial b_1} = 0.5 \qquad \frac{\partial L_i}{\partial w_2} = 7.5 \qquad \frac{\partial L_i}{\partial b_2} = 0.5$$

(c) Using a learning rate of 0.1, the next value for b_2 is given as

$$b_2 = b_2 - 0.1\cdot\frac{\partial L_i}{\partial b_2} = -15 - 0.05 = -15.05$$

3 Point Set Registration, 3 points

(a) Writing the equation system

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} a & 0 & -b \\ 0 & 1 & 0 \\ b & 0 & a \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix} \tag{1}$$

as a linear system in the variables a, b, t_x, t_y, and t_z yields

$$\begin{pmatrix} x & -z & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ z & x & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} a \\ b \\ t_x \\ t_y \\ t_z \end{pmatrix} = \begin{pmatrix} x' \\ y' - y \\ z \end{pmatrix} \quad .$$

One 3D-3D match yields 3 equations in 5 unknowns. We thus need 2 matches to obtain enough constraints to compute a transformation. We stack the resulting equations into a linear system $\mathtt{M}\theta = \mathbf{v}$, with $\theta = (a, b, t_x, t_y, t_z)^T$. Solving the linear system then provides an estimate of the parameters of the transformation.

(b) For $a^2 + b^2 = 1$, we can substitute $a = \cos\alpha$ and $b = \sin\alpha$ in Eq. 1. Eq. 1 thus represents a rotation around the y-axis (as the y value remains unchanged by the matrix), followed by a 3D translation.

(c) For matches $(x_1, y_1, z_1)^T \leftrightarrow (x'_1, y'_1, z'_1)^T$ and $(x_2, y_2, z_2)^T \leftrightarrow (x'_2, y'_2, z'_2)^T$, we obtain 6 equations in 5 unknowns. We thus can discard one equation for the solver, resulting in a 5×5 linear system for the solver that is minimal. For example, we could discard the equation

$$y'_2 = y_2 + t_y \quad .$$

If both matches are correct, then the discarded equation needs to be fulfilled. If one of the matches is incorrect (an outlier match), then the estimated transformation will be incorrect as well. In this case, there is a good chance that the discarded equation is not fulfilled. We can thus use the discarded equation to detect whether there is some problem with the estimated transformation: If the discarded equation is not fulfilled, we used at least one outlier to estimate it and the transformation will be incorrect. In this case, there is no need to evaluate the transformation on all matches.

Using this approach, RANSAC's run-time becomes:

$$t_{\text{RANSAC,new}} = n_{\text{good}} \left(t_{\text{solve}} + n \cdot t_{\text{verify}}\right) + n_{\text{bad}} \left(t_{\text{solve}} + t_{\text{verify}}\right) \quad ,$$

where n_{bad} and n_{good} are the number of iterations in which the random sample contains at least one outlier and only inliers, respectively. We have $n_{\text{bad}} + n_{\text{good}} = T$. Since we typically have $n_{\text{bad}} >> n_{\text{good}}$, we have $t_{\text{RANSAC}} >> t_{\text{RANSAC,new}}$.

4 RANSAC and Camera Pose Estimation, 3 points

Given n 2D-3D matches $\mathcal{M} = \{(\mathbf{x}_i, \mathbf{X}_i)\}$ between 2D image coordinates $\mathbf{x}_i$ and 3D point positions $\mathbf{X}_i$, camera pose estimation is the task of estimating the pose $\mathtt{R}$, $\mathbf{t}$ of the camera, *i.e.*, of computing the rotation $\mathtt{R}$ and translation $\mathbf{t}$ that best explains the matches.

(a) Pseudo-code for RANSAC-based pose estimation:

```
R, t = ransac_pose_estimation(M, K, tau)
    ε = 3/n    // initialize inlier ratio
    max_inliers = 0    // initialize max. number of inliers
    k_max = log(0.01)/log(1-ε^3)    // initialize max. number of iteration
    for t = 1 : k_max
        M' = select 3 matches at random from M
        R, t = minimal_solver(M')
        inliers = 0
        for i = 1 : n
            error = reprojection_error(x_i, X_i, K, R, t)
            if error < tau
                inliers = inliers + 1
        if inliers > max_inliers
            max_inliers = inliers
            ε = max_inliers/n
            k_max = log(0.01)/log(1-ε^3)
            R_best = R
            t_best = t
    Return R_best, t_best
```

(b) Given the pose $\mathtt{R}$, $\mathbf{t}$ and the intrinsic camera calibration matrix $\mathtt{K}$, the projection matrix for the camera is defined as

$$\mathtt{P} = \mathtt{K}[\mathtt{R}|\mathbf{t}] \in \mathbb{R}^{3\times 4} \quad .$$

Let $\mathbf{P}_i \in \mathbb{R}^{1\times 4}$ be the i-th row of the projection matrix. Furthermore, let $\mathbf{x}_i = (x_i, y_i)^T$ be the 2D coordinate of the 2D-3D match $(\mathbf{x}_i, \mathbf{X}_i)$.

We use the projection matrix to obtain the projection of the 3D point into the image in homogeneous coordinates as

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \quad = \quad \mathtt{P} \begin{pmatrix} \mathbf{X}_i \\ 1 \end{pmatrix},$$

where $x = \mathbf{P}_1 \begin{pmatrix} \mathbf{X}_i \\ 1 \end{pmatrix}$, $y = \mathbf{P}_2 \begin{pmatrix} \mathbf{X}_i \\ 1 \end{pmatrix}$, and $z = \mathbf{P}_3 \begin{pmatrix} \mathbf{X}_i \\ 1 \end{pmatrix}$. The reprojection error is then defined as

$$\text{error} = \begin{cases} \left\| \frac{1}{z} \begin{pmatrix} x \\ y \end{pmatrix} - \begin{pmatrix} x_i \\ y_i \end{pmatrix} \right\| & \text{if } z > 0 \\ \infty & \text{otherwise} \end{cases} .$$

The second case corresponds to the scenario where the 3D point is not in front of the camera. In this case, the reprojection error is not defined. In the case that the point is in front of the

camera, the reprojection error corresponds to the Euclidean distance between the measured 2D image position $\mathbf{x}_i$ and the 2D coordinate obtained by projecting the 3D point $\mathbf{X}_i$ into the image.

(c) The probability of drawing an all-inlier sample in a RANSAC iteration is given as ε^m. Thus, the probability of selecting at least one outlier in the sample is given by $1 - \varepsilon^m$.

Consequently, the probability of drawing at least one outlier (=not sampling an all-inlier sample) in each of k RANSAC iterations is given by $(1 - \varepsilon^m)^k$.

We are interested in obtaining $k_{\max}$ such that

$$(1 - \varepsilon^m)^{k_{\max}} \leq \tau \ .$$

Taking the logarithm on both sides of the inequality yields

$$k_{\max} \cdot \log(1 - \varepsilon^m) \leq \log(\tau) \ ,$$

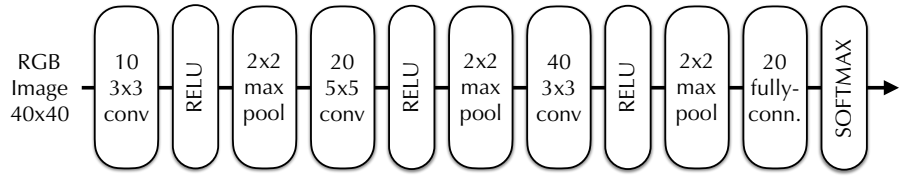
resulting in

$$k_{\max} \geq \frac{\log(\tau)}{\log(1 - \varepsilon^m)} \ ,$$

where the change in the direction of the inequality comes from the fact that $(1 - \varepsilon^m) < 1$ (unless $\varepsilon = 1$, where a single RANSAC iteration is sufficient) and thus $\log(1 - \varepsilon^m) < 0$. In practice, we thus obtain the number of required RANSAC iterations as

$$k_{\max} = \max\left(\left\lceil \frac{\log(\tau)}{\log(1 - \varepsilon^m)} \right\rceil, 1\right). \tag{2}$$

5 Deep Learning, 3 points



(a) The first conv. block uses 10 convolutions with $3 \times 3 \times 3 + 1$ parameters each, due to 1 bias term and a 3-dimensional (RGB) input. Thus, the first conv. block has $10 \cdot (3 \times 3 \times 3 + 1) = 280$ parameters.
The second conv. block uses 20 convolutions with $5 \times 5 \times 10 + 1$ parameters each. Thus, the second conv. block has $20 \cdot (5 \times 5 \times 10 + 1) = 5020$ parameters.
The third conv. block uses 40 convolutions with $3 \times 3 \times 20 + 1$ parameters each. Thus, the third conv. block has $40 \cdot (3 \times 3 \times 20 + 1) = 7240$ parameters.
The input to the FC layer is of size $5 \times 5 \times 40$ due to the use of padding and the pooling layers. Thus, accounting for bias terms, the FC layer has $20 \times (5 \times 5 \times 40) + 20 = 20 \times (25 \times 40) + 20 = 20020$ parameters.
All other layers do not have trainable parameters.

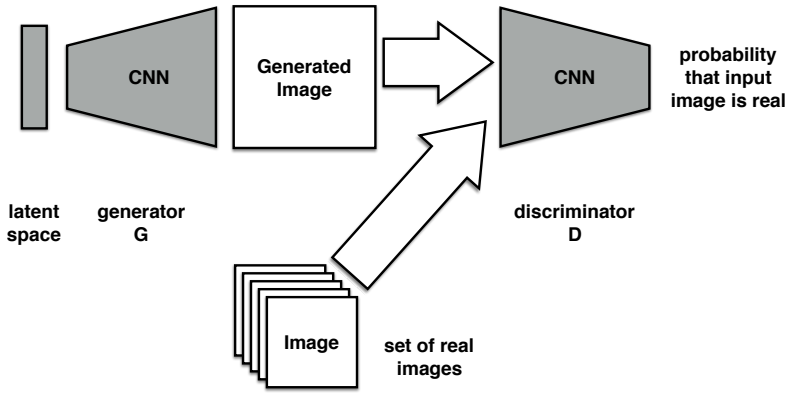
(b) Overfitting occurs if the number of trainable parameters is large compared to the size of the training dataset. The idea behind transfer learning is to first pre-train a network on a larger dataset with labels for a different task. After training, the layers specific to the training set and task are replaced with layers for our actual task and dataset. For example, the fully connected layers in a VGG network pre-trained on ImageNet are responsible for classifying an image into one of the 1000 classes of the ImageNet dataset. If we want to classify images into cats and dogs, we replace these layers with layers for binary classification. We then fix some of the convolutional layers pre-trained on the larger dataset to reduce the number of trainable parameters and train the network on our actual dataset.

The motivation behind transfer learning is that the first few layers of a network typically represent generic features such as corners and edges that are suitable for a wide range of tasks. As such, it should be sufficient to re-use the features from the pre-trained network. This results in fewer trainable parameters and thus reduces the risk of overfitting.

(c) "Data augmentation" enlarges the set of training examples by randomly rotating, scaling, translating, cropping, etc. the images and adding noise or brightness changes. The modifications should be small enough that they do not change the nature of the training images, i.e., that we can use the labels of the original images for the modified images. This increases the number of training samples and the variety of the training set. This can help to prevent overfitting to a too small training set.

6 Generative Adversarial Networks, 3 points

(a) The general architecture of a GAN is



Given a random sample $z \in \mathbb{R}^n$ from the latent space (drawn based on a selected probability distribution) as input, the generator G generates an image $G(z)$. Given either a generated image $G(z)$ or a real image x as input, the discriminator D outputs the probability $D(y)$ that the input image y is a real image. Here, $D(y) = 1$ corresponds to the case that the disciminator considers the image as real while $D(y) = 0$ corresponds to the case that the discriminator classifies the image as "fake". The discriminator and generator are typically implemented as convolutional neural networks (CNNs).

(b) The training objective of a GAN is given by

$$\min_G \max_D E_{\mathbf{x}\sim p_{\text{data}}}[\log(D(\mathbf{x}))] + E_{\mathbf{z}\sim p_{\text{model}}}[\log(1 - D(G(\mathbf{z})))] \ . \tag{3}$$

Here, D and G denote the discriminator and generator, respectively. $D(\mathbf{x})$ and $D(G(\mathbf{z}))$ are the probabilities that a real image $\mathbf{x}$ and a generated image $G(\mathbf{z})$ are considered a real image by the discriminator. Consequently, the discriminator aims at maximizing the probability $D(\mathbf{x})$ that a real image is correctly recognized as real while minimizing the probability $D(G(\mathbf{z}))$ (=maximizing the probability $1 - D(G(\mathbf{z}))$) that a generated image is falsely recognized as real. In contrast, the generator aims at generating realistically-looking images that are able to "fool" the discriminator. Consequently, the generator aims to maximize the probability $D(G(\mathbf{z}))$ (=minimizing the probability $1 - D(G(\mathbf{z}))$) that a generated image is considered as real by the discriminator.

(c) The gradients used to update the generator depend on the discriminator. If the discriminator perfectly distinguishes between fake and real images, the gradients for the generator vanish and the generator cannot be improved further.

If the discriminator is too easily fooled, the gradients for the generator vanish as well and there is no further improvement in the quality of the generated images.