Autonomous Vision Group
Prof. Dr.-Ing. Andreas Geiger

Eberhard Karls
Universität
Tübingen

Computer Vision

Exercise 6 – Self-Supervision and Diverse Topics

1 Pen and Paper

1.1 Self-Supervised Optical Flow and Depth

a) You are given the following sequence of 5×7 images, where part of the background is occluded by a thin structure (represented by the column of 10s). The sequence has 4 frames, denoted by t=1 to t=3.

1	2	3	10	5	6	7
1	2	3	10	5	6	7
1	2	3	10	5	6	7
1	2	3	10	5	6	7
1	2	3	10	5	6	7

(a) t=1

1	2	3	10	5	6	7
1	2	3	10	5	6	7
1	2	3	10	5	6	7
1	2	3	10	5	6	7
1	2	3	10	5	6	7

(b) t=2

1	2	10	4	5	6	7
1	2	10	4	5	6	7
1	2	10	4	5	6	7
1	2	10	4	5	6	7
1	2	10	4	5	6	7

(c) t=3

2	10	4	5	6	7	8
2	10	4	5	6	7	8
2	10	4	5	6	7	8
2	10	4	5	6	7	8
2	10	4	5	6	7	8

(d) t=4

Try to annotate the 5×7 optical flow labels in the x and y directions for the three frame transitions in the video sequence (t=1 to t=2, t=2 to t=3 and t=3 to t=4). You can assume zero padding at the image boundaries where necessary. Record the total time you spent labeling these images with a stopwatch. Assuming you can annotate a real video with the same speed, approximate the time required to label the optical flow for a 10 second video sequence of frame-rate 30 fps and resolution 1024×1024 pixels. Based on your findings, do you think it is feasible to manually annotate optical flow datasets for training deep networks?

For all 3 frame transitions, the optical flow in the y direction is zero at all pixels, since the only motion observed is horizontal. In addition, the flow along the x direction is also zero for the transition from t=1 to t=2, as these images are identical.

Optical Flow in x-direction for t=2 to t=3

0	0	-1	-1	0	0	0
0	0	-1	-1	0	0	0
0	0	-1	-1	0	0	0
0	0	-1	-1	0	0	0
0	0	-1	-1	0	0	0

Note how it is non-trivial to find the correspondences for the third column (the five pixels with intensity '3' at t=2), since there exists no exact correspondence in the frame at t=3. Here we choose the intensity '2' as the correspondence for this column, which leads to an optical flow field that is smooth. When dealing with ambiguous correspondences, a prior assumption of smoothness is therefore useful.

Optical Flow in x-direction for t=3 to t=4

-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1

Here, the flow of the left image boundary is ambiguous, as the column of '1's has no exact correspondence at t=4. A useful idea is to estimate the backward flow from t=4 to t=3 instead, and invert it. In this case, the left-most column becomes the column of '2's, for which there exists a corresponding column at t=3. This is called the forward-backward consistency assumption. Note that we use the backwards flow only for the left-most column, the forward flow can be computed for the rest of the image.

Assuming you took 1 minute to annotate these $5 \times 7 \times 4 = 140$ pixels, the time to annotate $1024 * 1024 * 300$ pixels would be 2246948 minutes or 4.27 years. It is also far harder to find correspondences in real videos than these toy examples. This means manually annotating optical flow datasets for deep neural networks is infeasible. The alternatives that are commonly used are (i) annotation with algorithms or additional sensors, (ii) rendering synthetic datasets for training, and (iii) self-supervised optical flow estimation, as we discussed in the lecture.

b) Do you think it is harder to perform self-supervised optical flow and depth estimation for images of a vehicle with well-polished, shiny paint or one with a dull painted surface? Explain your reasoning.

Dull surfaces should be simpler than shiny ones for self-supervised optical flow and depth estimation. This is because these approaches rely on photoconsistency loss terms, which assume that the image pixel intensity of the same 3D point does not change between camera views. Shiny surfaces are specular, so the intensity of light they reflect can change significantly from different viewpoints, breaking this assumption.

c) In the stereo lecture, we discussed the benefits of left-right consistency checks. Similarly, it is common in optical flow networks to perform bi-directional training, where the network estimates both the forward and backward optical flow. Does a forward-backward consistency loss for optical flow have the same purpose as a left-right consistency loss in stereo models? If not, discuss the differences.

The forward-backward consistency assumption in optical flow is that for non-occluded pixels in the first frame, the forward flow must be the inverse of the backward flow at the corresponding pixel in the second frame. This is used in two ways, similar to left-right consistency checks in stereo depth estimates:

i) to regularize training by ensuring coherence between the forward and backward flow

ii) to remove outliers and occlusions by masking out pixels with large inconsistencies

d) Self-supervised monocular depth estimation recovers the disparity map for the input image (where the units are in pixels). Is it possible to convert this into an actual depth map with physical units (meters)? If so, how would this be done?

Disparity can be converted to depth by the equation

$$depth = baseline * focal\ length/disparity \tag{1}$$

Therefore, depth and disparity are inversely related. There are 2 methods to approximate the constant that relates depth and disparity:

i) Since the monocular estimation network is trained on a specific data distribution, the disparity predicted will be based on the baseline and focal length used for collecting the training dataset. It is possible to directly use these values if they are known. In practice, most datasets used for depth estimation tasks provide these quantities.

ii) The value $baseline * focal\ length$ can also be estimated by imaging an object at known depth and estimating its disparity with the network to get a (depth, disparity) correspondence that can be substituted into Eq. (1).

e) The training objective (loss function) does not change for self-supervised depth prediction when using stereo images rather than nearby frames in a video sequence. However, stereo data requires a specialized camera setup to collect. Given this, mention two key advantages of stereo-based training.

i) Since stereo images are taken at the same time instant, we can avoid the challenge of handling moving objects.

ii) The relative pose between a pair of stereo images is known, and need not be estimated by the algorithm during training.

iii) The warping operation can be constrained to only the same horizontal image row, simplifying learning.

1.2 Pretext Tasks

a) A key problem in designing pretext tasks for self-supervised learning is the tendency of the network to prioritize trivial shortcuts over meaningful learning. Considering the approach discussed in the lecture of learning by solving jigsaw puzzles, what are the three main shortcuts exploited by the network, and the solutions proposed to overcome these?

i) Low level statistics: Adjacent patches include similar low-level statistics (mean and variance). Solution: normalize patch mean and variance.

ii) Edge continuity: A strong cue to solve Jigsaw puzzles is the continuity of edges. Solution: select 64×64 pixel tiles randomly from 85×85 pixel cells.

iii) Chromatic Aberration: Chromatic aberration is a relative spatial shift between color channels that increases from the images center to the borders. Solution: use grayscale images or spatial jitter each color channel by few pixels.

b) Consider the following two pretext tasks: (i) predicting if an image has been horizontally flipped, (ii) predicting if an image has been vertically flipped. When trained on a dataset of images from the internet, which of the two pretext tasks would you expect to lead to better self-supervised representations, and why?

Predicting if a natural image has been horizontally flipped is an ill-posed problem, with the exception of certain specific kinds of images (eg. images containing text). For most natural images, there should be no underlying feature that can be used to determine if the image has been mirrored. If this pretext task can be learned by a self-supervised model, it is likely that the model is exploiting some shortcut, and therefore the representations learned are less likely to be useful.

On the other hand, predicting vertical flips is a reasonable pretext task for images from the internet. A model that can accurately predict if an image has been flipped vertically based on whether objects in the image are in their 'canonical pose' (i.e., using relative position cues of what is usually at the top and bottom of an object). Features learned for this task could generalize to other image-based tasks.

c) If the same two pretext tasks from the previous question were applied to satellite images (eg., `https://earthview.withgoogle.com/`), which of the two would you now expect to lead to better representation learning?

Satellite images typically do not have strong spatial biases/structure like images from the internet. The images have a lot of texture and not many well-defined objects. Furthermore, the objects do not have a 'canonical pose'. Both kinds of image flipping are therefore unlikely to lead to useful representations for satellite images, since the problem is again ill-posed.

1.3 Contrastive Learning

a) An important design choice in nearly all contrastive learning methods is the score function or similarity metric. Given two features $\mathbf{f}_1$ and $\mathbf{f}_2$, write down the mathematical equation for the following 3 commonly used score functions: (i) L_1 (Manhattan) distance, (ii) L_2 (Euclidean) distance, and

(iii) cosine similarity. Rank the three functions based on how susceptible you think they are to outliers when used in high-dimensional feature spaces.

Remark: *An outlier here refers to an example for which the features lie far away from the mean of the data distribution.*

Let $\mathbf{f}_k^i$ denote the i^{th} dimension of the vector $\mathbf{f}_k$.

L_1 (Manhattan) distance: $s(\mathbf{f}_1, \mathbf{f}_2) = -||\mathbf{f}_1 - \mathbf{f}_2||_1 = -\sum_i |(\mathbf{f}_1^i - \mathbf{f}_2^i)|$

L_2 (Euclidean) distance: $s(\mathbf{f}_1, \mathbf{f}_2) = -||\mathbf{f}_1 - \mathbf{f}_2||_2 = -\sqrt{\left(\sum_i (\mathbf{f}_1^i - \mathbf{f}_2^i)^2\right)}$

Cosine similarity: $s(\mathbf{f}_1, \mathbf{f}_2) = \frac{\mathbf{f}_1^\top \mathbf{f}_2}{||\mathbf{f}_1||_2 \; ||\mathbf{f}_2||_2}$

Intuitively, since the L_2 distance squares the error (increasing it by a lot along dimensions with error > 1), the model will see a much larger error than the L_1 distance. The model becomes more sensitive to examples with large errors, and adjusts the parameters to minimize this error. If this example is an outlier, the model will be adjusted to minimize this single outlier case, at the expense of many other common examples, since the errors of these common examples are small compared to that single outlier case. On the other hand, cosine similarity measures only the angle between two vectors and is invariant to their magnitude (which is normalized in the denominator). Therefore, it is the least impacted to outliers. The ranking of functions is:

Cosine similarity $<$ L_1 distance $<$ L_2 distance

b) The InfoNCE loss for 1 reference (x), 1 positive (x^+) and N-1 negative (x_j^-) examples is given as follows:

$$\mathcal{L} = -\mathbb{E}_X \left[\log \frac{\exp(s(f(x), f(x^+)))}{\exp(s(f(x), f(x^+))) + \sum_{j=1}^{N-1} \exp(s(f(x), f(x_j^-)))} \right]$$

Consider an alternate loss function where there are no negative examples, such that the denominator is removed from the InfoNCE loss as follows:

$$\mathcal{L} = -\mathbb{E}_X \left[s(f(x), f(x^+)) \right]$$

What is the problem with this loss function in practice, which leads to the need for negative examples during training?

There exists a trivial solution to minimize this alternate loss function $s(f(x), f(x^+))$, which is to simply output $f(x) = 0$ for all x. If $f(x)$ is always zero, the InfoNCE loss becomes $\log N$, a relatively high value. However, if the second term in the denominator of the InfoNCE loss becomes zero, the total loss also becomes zero. In order to do this, $s(f(x), f(x_j^-))$ must become a large negative value. Therefore, the InfoNCE loss avoids the trivial solution by encouraging $f(x)$ and $f(x_j^-)$ to be different from each other.

1.4 Input Optimization

a) Explain the difference between white-box and black-box adversarial attacks. Which of the two types of attacks is harder to achieve in practice?

In white-box attacks the attacker has access to the model's parameters, while in black-box attacks, the attacker has no access to these parameters, i.e., it tries to generate adversarial images with the hope that they are robust and will transfer to the target model. Black-box attacks are harder to achieve in practice.

b) Consider the following 2×2 image:

$$\begin{bmatrix} \mathbf{a} & \mathbf{b} \\ \mathbf{c} & \mathbf{d} \end{bmatrix}$$

We apply two element-wise feature extractors, F_1 and F_2, to obtain the following feature channels:

$$\begin{bmatrix} F_1(\mathbf{a}) & F_1(\mathbf{b}) \\ F_1(\mathbf{c}) & F_1(\mathbf{d}) \end{bmatrix}, \begin{bmatrix} F_2(\mathbf{a}) & F_2(\mathbf{b}) \\ F_2(\mathbf{c}) & F_2(\mathbf{d}) \end{bmatrix}$$

In style transfer applications, the style of an image is expressed as a Gram Matrix:

$$G_{ij} = \sum_{\mathbf{p}\in\Omega} F_i(\mathbf{p})F_j(\mathbf{p})$$

Express the 2×2 Gram Matrix G for these two feature channels in terms of F_1, F_2, $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$ and $\mathbf{d}$.

Hint: *Flatten the image into a 4-dimensional vector and represent the feature map as a* 2×4 *matrix before computing the Gram Matrix.*

The feature map can be represented as:

$$\begin{bmatrix} F_1(\mathbf{a}) & F_1(\mathbf{b}) & F_1(\mathbf{c}) & F_1(\mathbf{d}) \\ F_2(\mathbf{a}) & F_2(\mathbf{b}) & F_2(\mathbf{c}) & F_2(\mathbf{d}) \end{bmatrix}$$

The gram matrix for this feature map is:

$$\begin{bmatrix} F_1(\mathbf{a})^2 + F_1(\mathbf{b})^2 + F_1(\mathbf{c})^2 + F_1(\mathbf{d})^2 & F_1(\mathbf{a})F_2(\mathbf{a}) + F_1(\mathbf{b})F_2(\mathbf{b}) + F_1(\mathbf{c})F_2(\mathbf{c}) + F_1(\mathbf{d})F_2(\mathbf{d}) \\ F_2(\mathbf{a})F_1(\mathbf{a}) + F_2(\mathbf{b})F_1(\mathbf{b}) + F_2(\mathbf{c})F_1(\mathbf{c}) + F_2(\mathbf{d})F_1(\mathbf{d}) & F_2(\mathbf{a})^2 + F_2(\mathbf{b})^2 + F_2(\mathbf{c})^2 + F_2(\mathbf{d})^2 \end{bmatrix}$$

2 Coding Exercises

The coding exercises use the jupyter notebook `code/self_supervision.ipynb`. You will implement a k-Nearest Neighbor classifier for CIFAR-10 using self-supervised representations from the *Barlow Twins* approach discussed in the lecture. As in the previous exercises, the notebook is self-contained. The dataset will be downloaded to your working environment as you execute the functions in the notebook.

When you see helper functions in the notebook, you don't need to do anything– they are already implemented. The functions you need to implement are indicated as "Exercise Function". For this exercise, there are **four exercise functions** in total. When you are done, you can explore the approach in more detail if you like, by re-training the model with different augmentations and hyper-parameters.