

METHOD FOR TRAINING A NEURAL NETWORK

TECHNICAL FIELD

[0001] Various aspects of this disclosure relate to methods for training a neural network.

BACKGROUND

[0002] Self-supervised learning (SSL) aims to train a highly transferable deep model (i.e. a neural network) on unlabeled data by solving a well-designed pretext task which can generate pseudo targets for the task itself.

[0003] Autoencoders may be used to learn efficient encoding of data in a self-supervised manner. One option for efficiently training an encoder for a computer vision (downstream) task, e.g. a classification, object detection or segmentation task, is the masked autoencoder (MAE) approach.

[0004] For a pre-training phase, according to MAE, an input image where patches are randomly masked is fed into the encoder and the autoencoder is trained such that its decoder can reconstruct the pixels or features of the masked patches from the latent representation generated by the encoder and mask tokens (i.e. information about which patches are masked). After pre-training, the encoder is fine-tuned for the downstream task via standard supervised training.

[0005] One can observe that the core of the MAE framework is the masking on the encoder input, which unfortunately causes inconsistency between the pre-training and fine-tuning phases. Specifically, for the encoder, the input is a masked or incomplete one in the pre-training phase, while it is complete without masking in the fine-tuning phase. This inconsistency may impair the performance. Moreover, though being well compatible with the vision transformers (ViT) encoder, the masking strategy employed on the encoder input according to MAE prohibits the pre-training of other popular and effective encoder architectures, e.g. CNN (convolutional neural network), MLP(multi-layer perceptron)-based architectures, or others. This is because these popular architectures cannot handle incomplete input due to convolutions and pooling operations in CNNs and fully-connected layers in MLP-based architectures.

[0006] Accordingly, approaches for training a neural network are desirable which achieve good results without masking of the encoder input.

SUMMARY

[0007] Various embodiments concern a method for training a neural network, including forming an autoencoder including the neural network as encoder and including a decoder, for each training image of multiple training images, generating a latent representation of the training image by the encoder, transforming the training image and supplying information about the transformation and at least a part of the latent representation to the decoder to generate a decoder output for the training image and adjusting the encoder and the decoder to reduce a loss between the transformed training images and the decoder outputs.

[0008] According to one embodiment, the method includes masking the latent representation and supplying the masked latent representation to the decoder to generate the decoder output.

[0009] According to one embodiment, the method includes subdividing the training image into a plurality of training image patches, wherein the latent representation includes an encoding for each training image patch and wherein masking the latent representation includes replacing at least some of the training image patches by mask tokens.

[0010] The image may be subdivided into the patches according to a regular pattern. In particular, the patches may all be of the same size. For example, an input image of size 224x224 pixels is divided into 14x14 patches of size 16x16 pixels.

[0011] According to one embodiment, the method includes randomly selecting the training image patches replaced by mask tokens.

[0012] According to one embodiment, the method includes adjusting the mask tokens and the encoder and the decoder to reduce the loss between the transformed training images and the decoder outputs.

[0013] According to one embodiment, the loss between the transformed training images and the decoder outputs includes a mean-square-error loss, a cosine distance or a Kullback-Leibler divergence of the transformed training images and the decoder outputs.

[0014] According to one embodiment, the transformation includes a feature extraction of the training image followed by a homography transformation.

[0015] According to one embodiment, the transformation is a homography transformation of the training image.

[0016] According to one embodiment, the information about the transformation is an encoding of hyper parameters of the transformation.

[0017] According to one embodiment, the method includes generating the encoding of hyper parameters of the transformation by a further neural network.

[0018] According to one embodiment, the method includes adjusting the further neural network and the encoder and the decoder to reduce the loss between the transformed training images and the decoder outputs.

[0019] According to one embodiment, the neural network is a convolutional neural network a vision transformer network or a multi-layer perceptron-based neural network. Other types of neural networks for computer vision may also be used, i.e. the method is flexible with regard to the structure of the encoder.

[0020] According to one embodiment, a training device is provided configured to perform the method for training a neural network as described above.

[0021] According to one embodiment, a computer program element is provided including program instructions, which, when executed by one or more processors, cause the one or more processors to perform the method for training a neural network as described above.

[0022] According to one embodiment, a computer-readable medium is provided including program instructions, which, when executed by one or more processors, cause the one or more processors to perform the method for training a neural network as described above.

[0023] It should be noted that embodiments described in context of the method are analogously valid for the device.

BRIEF DESCRIPTION OF THE DRAWINGS

[0024] The invention will be better understood with reference to the detailed description when considered in conjunction with the non-limiting examples and the accompanying drawings, in which:

- FIG. 1 shows an autoencoder.
- FIG. 2 illustrates a masked autoencoder approach for training a neural network.
- FIG. 3 illustrates the TAE (Transformed Autoencoder) approach for training a neural network.

- FIG. 4 illustrates the partitioning of an encoder input into non-overlapping patches to tokenize them into a series of patch tokens which are then fed into a decoder to obtain a series of pixels for predicting a spatially transformed target.
- FIG. 5 shows a flow diagram illustrating a method for training a neural network.

DETAILED DESCRIPTION

[0025] The following detailed description refers to the accompanying drawings that show, by way of illustration, specific details and embodiments in which the disclosure may be practiced. These embodiments are described in sufficient detail to enable those skilled in the art to practice the disclosure. Other embodiments may be utilized and structural, and logical changes may be made without departing from the scope of the disclosure. The various embodiments are not necessarily mutually exclusive, as some embodiments can be combined with one or more other embodiments to form new embodiments.

[0026] Embodiments described in the context of a device are analogously valid for a method and vice-versa.

[0027] Features that are described in the context of an embodiment may correspondingly be applicable to the same or similar features in the other embodiments. Features that are described in the context of an embodiment may correspondingly be applicable to the other embodiments, even if not explicitly described in these other embodiments. Furthermore, additions and/or combinations and/or alternatives as described for a feature in the context of an embodiment may correspondingly be applicable to the same or similar feature in the other embodiments.

[0028] In the context of various embodiments, the articles “a”, “an” and “the” as used with regard to a feature or element include a reference to one or more of the features or elements.

[0029] As used herein, the term “and/or” includes any and all combinations of one or more of the associated listed items.

[0030] In the following, embodiments will be described in detail.

[0031] FIG. 1 shows an autoencoder 100.

[0032] The autoencoder 100 includes an encoder 101 and a decoder 102 (both implemented by a respective neural network).

[0033] The encoder 101 encodes an input 103 to a latent representation 104 in a latent space and the decoder 102 generates an output from a latent representation.

[0034] According to the MAE (Masked Autoencoder) approach (more generally a masked image modelling SSL approach), the input 103, which is in that case an image, is masked before it is fed into the encoder 101, as illustrated in FIG. 2.

[0035] FIG. 2 illustrates a masked autoencoder approach.

[0036] In this approach, the encoder 201 receives as input 203 an image in which patches are randomly masked, i.e. the encoder 201 receives as input 203 only the patches of the image which are not masked. The encoder 201 encodes each unmasked patch to a respective encoded patch. The encoded patches form together the latent representation 204. The decoder 202 is to generate an output 205 by reconstructing the pixel values of the masked patches from the latent representation of the encoder and mask tokens 206 (which take the place of encodings for patches which have been masked). This mask reconstruction pretext task (for pre-training of the neural network taking the place of the encoder 201) is also denoted as masked image modelling. Given a specific downstream task, this SSL family fine-tunes the pre-trained encoder on the corresponding training data in a supervised manner.

[0037] However, the fact that for the encoder the input is masked for pre-training and is not masked for fine-tuning causes inconsistency in training and may impair performance. Further, encoder architectures like CNNs and MLP-based architectures are not compatible with the masking strategy on the encoder input, which limits wider application of the MAE-like SSL family.

[0038] In view of the above, according to various embodiments, a training approach is provided which is mask-free at the encoder input and thus avoids the above two issues. The approach provided, which is denoted as TAE (Transformed Autoencoder) approach, uses transformed image modelling to reconstruct the input image or its semantic feature.

[0039] FIG. 3 illustrates the TAE approach for mask-free encoder (i.e. without mask at the encoder's input) pre-training.

[0040] According to the TAE approach, an encoder 301, denoted as f for the encoding function it implements, is used to encode a (full, i.e. unmasked) input image 203 (which may be a crop of a larger image), denoted as x in the following, into a set of latent patch tokens, i.e. a latent representation 304, denoted as z , including an encoding (encoding token, also referred to as latent patch token) for each of a plurality of patches into which the input image

203 is subdivided. The encoder 301 is trained together with the decoder 302 such that the decoder 302, denoted as g , recovers a spatially transformed version $\mathcal{T}_s(\mathbf{x})$ of the input $\mathbf{x}$ from the latent representation 304 in which encoding tokens are randomly masked (i.e. replaced by mask tokens, i.e. there are mask tokens at the masked positions within the latent representation) and an embedding 306 of parameters of the spatial transformation $\mathcal{T}_s$. The reconstruction target can, instead of the transformed version of the input image $\mathcal{T}_s(\mathbf{x})$ also be semantic features $\mathcal{T}_s(f'(\mathbf{x}))$ of the input image, where f' is, for example, the exponentially moving average of f .

[0041] In the following, a Vision Transformer (ViT) is used as an example as the encoder 301 but other architectures, such as CNN and MLP-based networks, can also be implemented as the encoder 301 in TAE. Accordingly, in the following, it is described how a ViT backbone network can be trained as an encoder within a TAE framework.

[0042] As mentioned above, the input image 303 is divided into a set of non-overlapping patches and these patches are fed into the encoder 301. The encoder 301, in this example a standard ViT network, uses a linear projection to generated latent space embeddings (or encodings) for the image patches, and then adopts a series of transformer blocks to process the patch embeddings with positional embeddings added at the beginning. In this way, the encoder 301 outputs a series of latent patch tokens, which form the latent representation 304.

[0043] The decoder 302 consists of a series of standard transformer blocks. In the following, it is described how the decoder 302 processes the latent patch tokens $\mathbf{z}$ given by the encoder 301.

[0044] A spatial image transformation $\mathcal{T}_s$ is selected from a set of possible transformations and values of hyper-parameters $\boldsymbol{\sigma}$ which specify the selected transformation among the possible transformations are encoded into the transformation embedding (or encoding) 306, denoted as $e_{\mathcal{T}_s}$ via a small, e.g. 2-layer, MLP according to

$$e_{\mathcal{T}_s} = \text{MLP}(\boldsymbol{\sigma}) \in \mathbb{R}^d, \quad (1)$$

where d denotes the dimension of the latent patch tokens $\mathbf{z}$. For example, to implement the spatial transformation $\mathcal{T}_s$ a homography transformation with eight degrees of freedom is used (the most general type of spatial transformations on 2D planes).

[0045] Then, the latent patch tokens $\mathbf{z}$ are randomly masked by replacing each one of (randomly) selected tokens with a shared and learned (i.e. trainable) mask token (as represented by the hatched tokens of the latent representation 304). Next, positional embeddings are added to all tokens in $\mathbf{z}$ to tell the locations of all patches (to which the tokens

correspond) in the (original) image $\mathbf{x}$. Finally, the hyper-parameter embedding $e_{\mathcal{T}_s}$ of

$e_{\mathcal{T}_s}$ is concatenated to each token (mask token or token generated by the encoder 301) so as to include into each token the information on what spatial transformation has been performed. Alternatively, $e_{\mathcal{T}_s}$ may be directly added to each token.

[0046] The result of this processing of the latent patch tokens $\mathbf{z}$ is fed to the decoder 302 to obtain a prediction 305, denoted as $\mathbf{y}'$.

[0047] Regarding the reconstruction target $\mathbf{y}$ for $\mathbf{y}'$, there are for example the options to

i) recover the spatially transformed pixels $\mathbf{y}$ of the image $\mathbf{x}$ and

ii) reconstruct a spatially transformed semantic feature $\mathbf{y}$ of the image $\mathbf{x}$,

i.e. $\mathbf{y}$ is given by

$$\mathbf{y} = \begin{cases} \mathcal{T}_s(\mathbf{x}), \\ \mathcal{T}_s(f'(\mathbf{x})), \end{cases} \quad (2)$$

wherein the upper option corresponds to the target being pixel reconstruction and the lower option corresponds to the target being feature reconstruction.

[0048] Here, f' is the exponentially moving average of f .

[0049] With the target $\mathbf{y}$ as for example above, the training loss of TAE may be defined as follows:

$$\min_{f,g} \frac{1}{|\mathcal{K}|} \sum_{i \in \mathcal{K}} \ell(\mathbf{y}_i, \mathbf{y}'_i), \quad (3)$$

where $\mathbf{y}_i$ denotes the component for the i -th patch in $\mathbf{y}$, $\mathcal{K}$ denotes a set consisting of the position indexes of patches of the input image $\mathbf{x}$.

[0050] Here the loss function ℓ measures the discrepancy between the prediction $\mathbf{y}'_i$ and the ground truth $\mathbf{y}_i$, e.g. the mean-square-error (MSE), cosine distance or KL Divergence.

[0051] The encoder 301 and the decoder 303 may be pre-trained on a large-scale unlabeled dataset (e.g. for 300 epochs on RGB images), and then take the encoder 301 as a feature extractor with or without fine-tuning (e.g. for 200 epochs) on other labelled datasets (e.g. on RGB images). The training data may include images of ImageNet-1k.

[0052] The spatial transformation $\mathcal{T}_s$ on the reconstruction target can be seen as a key component in TAE. It helps the encoder 301 to better learn the dependency among different patches in an image and also enhances data semantics learning.

[0053] FIG. 4 illustrates the partitioning of the encoder input into non-overlapping patches to tokenize them into a series of patch tokens $\mathbf{z}$, which are then fed into the decoder 302 to obtain a series of pixels $\mathbf{y}_i$ for predicting a spatially transformed target $\mathcal{T}_s(\mathbf{x})$. Since the encoder input $\mathbf{x}$ differs from the target $\mathcal{T}_s(\mathbf{x})$ due to the spatial transformation $\mathcal{T}_s$, the spatial partition for the patch tokens $\mathbf{z}$ in the encoder 301 and decoder 302 is different from the one in the target $\mathcal{T}_s(\mathbf{x})$. This means that there is no exact one-to-one correspondence between the patches based on which the encoder 301 generates the patch tokens $\mathbf{z}$ and those of the target $\mathcal{T}_s(\mathbf{x})$. Actually, as shown by the bold rectangle in the bottom right and the larger bold rectangle in the top right of FIG. 3, a patch in the transformed target can correspond (i.e. include information of) multiple patches in the input and, vice versa, the content of one token in $\mathbf{z}$ can be separated into (i.e. relate to) several nearby patches in $\mathcal{T}_s(\mathbf{x})$. However, to predict the corresponding patches in $\mathcal{T}_s(\mathbf{x})$, the patches of the

decoder prediction $\mathbf{y}'$ have a one-to-one correspondence with the patches in $\mathcal{T}_s(\mathbf{x})$. The prediction content of one token in $\mathbf{y}'$ thus actually comes from several nearby patch tokens in $\mathbf{z}$. Therefore, by the training, the TAE encoder 301 and the TAE decoder 302 are trained to exchange sufficient information among tokens for fusing several nearby token patches together to achieve small reconstruction loss. This accordingly induces patch dependency learning and also enhances learning of data semantics. Moreover, due to the masks on the decoder input, some of the necessary nearby tokens may be masked. This further boosts the encoder to exchange sufficient information among tokens such that each unmasked token in the decoder has contained enough information of other tokens and the decoder can use them to well predict the masked patches.

[0054] The random masking applied to the input of the decoder 302 further enhances the representation power of the features (i.e. encodings) learned in training, in addition to training the machine-learning model (i.e. the autoencoder) to be aware of spatial transformations. While in the approach illustrated in FIG. 2 image patches (e.g. of a Vision Transformer) is masked at the input of the encoder on $\mathbf{x}$ the model is to recover the masked regions, masking is performed after the encoder according to TAE on the latent representation $\mathbf{z}$, i.e. there is a masking operation $\mathbf{z}_m = \text{mask}(\mathbf{z})$ wherein a certain number of patch encodings is masked. For example, for 50% of the patches the encodings are masked (e.g. around 98 encodings are masked). By reconstructing all patches at the output of the decoder 302 from the masked latent representation $\mathbf{z}_m$, each (local, i.e. patch) encoding in $\mathbf{z}_m$ is trained to have a better representation of the whole input image 303. Unlike the spatial transformation that changes both the reconstruction target and the latent representation, the masking only changes the latent representation $\mathbf{z}$ and does not affect the reconstruction target.

[0055] As aforementioned, TAE does not mask the encoder input, and thus can be easily used to train other types of popular and effective architectures, including CNNs (e.g. ResNet [48]) and MLP-based networks (e.g. MLP-Mixers), etc. In principle, to pre-train such a non-ViT backbone with TAE, one can directly use the non-ViT backbone to implement the TAE encoder. But for a CNN and MLP-based backbone, one needs to remove its global pooling and fully connected layers at the end of the respective neural network (if there are any). Besides, for a CNN, e.g. ResNet, its output feature map is often of spatial-size 7×7 which is

much smaller than the input size 224×224 . To make the output feature map preserve more spatial details of the input image, a transposed convolution may be applied to the last stage, which may then be summed with the feature map from the second last stage to form a feature map of size 14×14 . For a MLP-Mixer, its latent patch tokens are the output of the last block like ViT without any special operation. For a TAE decoder, standard transformer blocks may be used to implement it for simplicity and consistency.

[0056] The decoder may be discarded after pre-training (i.e. before the fine-tuning phase).

[0057] In summary, according to various embodiments, a method is provided as illustrated in FIG. 5.

[0058] FIG. 5 shows a flow diagram 500 illustrating a method for training a neural network.

[0059] In 501 an autoencoder is formed including the neural network as encoder and including a decoder.

[0060] In 502, for each training image of multiple training images

- a latent representation of the training image is generated by the encoder in 503 (by applying it to, according to various embodiments, the unmasked training image);
- the training image is transformed in 504; and
- information about the transformation and at least a part of the latent representation are supplied to the decoder to generate a decoder output for the training image in 505.

[0061] In 506 the encoder and the decoder are adjusted to reduce a loss between the transformed training images and the decoder outputs.

[0062] The encoder for example determines the latent representation by determining patch-wise encodings of the training image (and, in inference, of the respective input image) of a subdivision of the training image into a plurality of patches. The transformation may for example be understood as a transformation which changes the association of pixels with patches, i.e. for each of at least some (not necessary all but a major part, e.g. 20%, 30% or 40%) of the pixels the pixel value of the pixel is shifted to another patch by the transformation.

[0063] The transformation is different for at least some of the training images, i.e., for example, parameters of the transformation differ between (at least some of) the training images.

[0064] For example, as mentioned above, each training picture may be crop of a larger, original image. According to various embodiments, each transformation is a transformation

on the original image such that the transformed crop (i.e. part of the image) takes contents from a region completely within the original image. More specifically, for example, a base crop is defined by the coordinates of its 4 vertices in the original image $p_0 = (x_{\min}, y_{\min})$, $p_1 = (x_{\max}, y_{\min})$, $p_2 = (x_{\max}, y_{\max})$, $p_3 = (x_{\min}, y_{\max})$. The scale of the crop is denoted as the length of its shorter side $s_x = \min(x_{\max} - x_{\min}, y_{\max} - y_{\min})$. Then for each vertex p_i , a new point p_i^t is randomly chosen within a small squared region of size λs_x centred around p_i . For example, by default, $\lambda = 0$. The corresponding region with the transformed vertices $p_0^t, p_1^t, p_2^t, p_3^t$ is then extracted from the original image, followed by resizing it to the training size image (e.g. 224×224 pixels) to form the transformed crop. The transformation parameters for this transformation are then obtained by calculating the perspective transformation matrix between the original coordinates p_0, p_1, p_2, p_3 to the new coordinates $p_0^t, p_1^t, p_2^t, p_3^t$. According to one embodiment, during pre-training, the probability of applying the spatial transform is linearly increased from 0 to 0.5. For image crops without the spatial transform, the same original crop is used as the reconstruction target.

[0065] The approach of FIG. 5, e.g. in form of the TAE approach detailed above, allows building an unsupervised pre-training framework applicable to general network architectures.

[0066] It thus achieves architecture compatibility and further allows achieving training consistency and orthogonality to other self-supervised learning (SSL) methods.

[0067] Firstly, with the mask-free (at the encoder input) encoder pre-training mechanism according to TAE, for both pre-training and fine-tuning phases, the full input image is fed into the encoder. In this way, for both phases, the TAE encoder always sees the whole picture of the input, and thus can consistently handle and learn the input patches. In contrast, for the MAE-like framework (illustrated in FIG. 2), the encoder input is masked in the pre-training phase but not masked in the fine-tuning phase, which indicates encoder training inconsistency between the two phases.

[0068] As mentioned above, the TAE encoder can be compatible to many popular and effective network architectures, including not only ViTs but also CNNs and MLP-based networks. This compatibility comes in particular from the mask-free strategy on the TAE encoder. In contrast, the MAE-like framework is often not suitable for non-ViT architectures and suffers from an architecture compatibility issue. This is because it cannot handle masked input due to convolutions and pooling operations in architectures like CNNs or spatial-MLP layers in MLP-based architectures.

[0069] Further, TAE with transformed image reconstruction is a very general framework and is compatible to many SSL families, such as MAE-like frameworks and contrastive learning methods. It can be combined with other SSL approaches to enjoy merits of both sides. Experimental results show that integrating the transformed image reconstruction task in TAE with the MAE-like framework, e.g. MAE, can improve their performance.

[0070] The methods described herein may be performed and the various processing or computation units and the devices and computing entities described herein may be implemented by one or more circuits. In an embodiment, a "circuit" may be understood as any kind of a logic implementing entity, which may be hardware, software, firmware, or any combination thereof. Thus, in an embodiment, a "circuit" may be a hard-wired logic circuit or a programmable logic circuit such as a programmable processor, e.g. a microprocessor. A "circuit" may also be software being implemented or executed by a processor, e.g. any kind of computer program, e.g. a computer program using a virtual machine code. Any other kind of implementation of the respective functions which are described herein may also be understood as a "circuit" in accordance with an alternative embodiment.

[0071] While the disclosure has been particularly shown and described with reference to specific embodiments, it should be understood by those skilled in the art that various changes in form and detail may be made therein without departing from the spirit and scope of the invention as defined by the appended claims. The scope of the invention is thus indicated by the appended claims and all changes which come within the meaning and range of equivalency of the claims are therefore intended to be embraced.

CLAIMS

1. A method for training a neural network, comprising:
Forming an autoencoder comprising the neural network as encoder and comprising a decoder;
For each training image of multiple training images
 Generating a latent representation of the training image by the encoder;
 Transforming the training image; and
 Supplying information about the transformation and at least a part of the latent representation to the decoder to generate a decoder output for the training image; and
Adjusting the encoder and the decoder to reduce a loss between the transformed training images and the decoder outputs.
2. The method of claim 1, comprising masking the latent representation and supplying the masked latent representation to the decoder to generate the decoder output.
3. The method of claim 2, comprising subdividing the training image into a plurality of training image patches, wherein the latent representation comprises an encoding for each training image patch and wherein masking the latent representation comprises replacing at least some of the training image patches by mask tokens.
4. The method of claim 3, comprising randomly selecting the training image patches replaced by mask tokens.
5. The method of claim 4, comprising adjusting the mask tokens and the encoder and the decoder to reduce the loss between the transformed training images and the decoder outputs.
6. The method of any one of claims 1 to 5, wherein the loss between the transformed training images and the decoder outputs comprises a mean-square-error loss, a cosine

distance or a Kullback-Leibler divergence of the transformed training images and the decoder outputs.

7. The method of any one of claims 1 to 6, wherein the transformation comprises a feature extraction of the training image followed by a homography transformation.
8. The method of any one of claims 1 to 6, wherein the transformation is a homography transformation of the training image.
9. The method of any one of claims 1 to 8, wherein the information about the transformation is an encoding of hyper parameters of the transformation.
10. The method of claim 9, comprising generating the encoding of hyper parameters of the transformation by a further neural network.
11. The method of claim 10, comprising adjusting the further neural network and the encoder and the decoder to reduce the loss between the transformed training images and the decoder outputs.
12. The method of any one of claims 1 to 11, wherein the neural network is a convolutional neural network, a vision transformer network or a multi-layer perceptron-based neural network.
13. A training device configured to perform the method of any one of claims 1 to 12.
14. A computer program element comprising program instructions, which, when executed by one or more processors, cause the one or more processors to perform the method of any one of claims 1 to 12.
15. A computer-readable medium comprising program instructions, which, when executed by one or more processors, cause the one or more processors to perform the method of any one of claims 1 to 12.