Deep learning mini-project

Deep learning for media technology (TNM112), lab 3

Abstract

The third lab is focused on more advanced applications of deep learning, going beyond the simpler classification tasks we have looked at in Lab 1 and 2. This lab is formulated as a more open assignment, or mini-project, where you are supposed to choose a topic from different suggestions or proposals, or come up with a topic yourself. You will apply the method and model that you select to a dataset of choice.

1 Introduction

There are three different proposals of what could be focused on in the lab, described in Section 4. These are to some extent focused on unsupervised learning and image-to-image CNNs, but there is also room to explore other directions of interest. While the suggestions here focus on images, you are also able to select some other data you find interesting to work with.

There is no code or pre-defined data provided for this lab. You are supposed to develop your model from scratch, or start from an available implementation of a model. Some suggestions of datasets are listed in Section 2, but you can also find some other dataset if you want. In Section 3, there is a description of what you are expected to achieve in the lab. In Section 4, three different proposals are described, where you can choose one of the proposals to focus on for the lab. You can also choose to work on something outside of the proposed topics, given that it is of a sufficient level of complexity.

2 Datasets

You are free to choose the dataset to work with. The easiest would probably be to use the data generator from Lab 2, where you can load either MNIST, CIFAR-10, or the PatchCamelyon dataset. However, there are many more datasets available online that you can choose from. As a starting point, you can have a look at: https://paperswithcode.com/datasets.

You should make your decision based on how much resources you are able to put in. With a high-resolution dataset you will need longer time for training, making development slower. You can also choose to go outside the imaging domain, e.g., selecting a dataset related to audio. However, you will have to spend more time thinking about how to represent your data. For example, you can represent an audio clip as an image by means of its spectrogram (or the Mel spectrogram to align better with sound perception).

3 Requirements

There are no strict requirements for passing the lab assignment. Suggestions of things to explore are provided in Section 4. Your mini-project should be on a level that it goes a bit beyond simply training an existing model on a dataset of choice. This could, e.g., be that you do some experiments with variations in the model, data, or optimization, and study the results based on some evaluation method. For example, you could use the autoencoder (AE) in Proposal 1 in Section 4.1 for pre-training, evaluating the results based on some downstream task, or you could evaluate the quality of the AE reconstruction for different dimensions of the bottleneck-/latent representation or other design choices.

There is also a difference between implementing a model from scratch compared to using an already implemented model in, e.g., Keras. For example, if you implement an image-to-image CNN from scratch for some task in Proposal 2 in Section 4.2, it is enough that you discuss your development in the lab report and provide some results

(such as examples of the model output and some measures of the performance with a suitable metric). However, if you take an already existing implementation and train it on a dataset of choice, you will need to go a bit deeper in evaluating the results based on different variations in terms of, e.g., data augmentations, optimization settings, etc.

Lab report: Your lab report is the main source for demonstrating your work, and thus for the examination. This should show with sufficient clarity the motivation of your work, describe the techniques you have used, and provide results that demonstrate how well your model works.

Code: Also provide all the source code required to run your model and experiments. If you have used some existing code, clearly specify what parts of the code is not yours.

3.1 Summary

In conclusion, we can formulate the following requirements of the lab assignment:

1. Implement one of the proposed models in Section 4, choose a different model of choice, or use an already implemented model.
2. Evaluate the performance of the model (numerical comparisons, such as MSE, and example images where appropriate). If you use an already implemented model, the evaluation should go a bit deeper, e.g. looking at the performance with different variations in model, data, and/or optimization.
3. Lab report discussing the motivation, method, and results, in a way that demonstrates your understanding of the topic.
4. Source code.

Requirements 1, 3, and 4 are quite straightforward, while requirement 2 is more open and depends on what model you choose in requirement 1.

4 Assignments

You can choose to focus your lab work on one of the three suggested areas provided in the proposals below. You are also free to come up with some other suggestion of what to focus on (see Section 4.4).

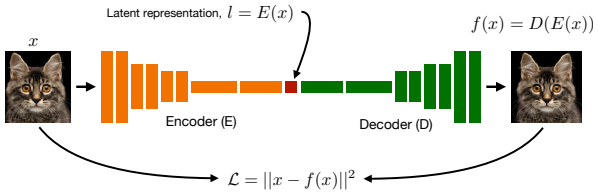

Figure 1: The autoencoder.

4.1 Proposal 1 – The autoencoder

In this proposal, you will focus on implementing an autoencoder (AE). This is composed of two distinct parts; the encoder and the decoder, see Figure 1. The encoder follows a similar design as the CNNs we looked at in Lab 2, with convolutional layers, pooling layers, flattening, and some fully connected/dense layers. The decoder is similar to a "mirrored" version of the encoder, starting with dense layers followed by a reshape layer (the opposite of the flattening layer) to structure the activations into channel representations. For example, if the output of a dense layer is 2048 activations, these could be reshaped into a 4×4×128 representation, i.e. 128 channels with resolution 4×4. For upsampling the layers (the "opposite" of a pooling layer) you can either use transposed convolutions, or do linear upsampling followed by convolutions. The relevant Keras layers to have a look at are:

```
keras.layers.Reshape(...)
keras.layers.Conv2DTranspose(...)
keras.layers.UpSampling2D(...)
```

The final layer of the AE should have the same resolution and number of channels as the the input image (e.g. 3 for RGB images). You can then formulate a loss function by comparing the output to the input images, e.g. using the L2 loss. You then train the AE unsupervised, i.e. you only need the images and not labels/annotations.

When you have implemented the AE, you can explore different directions for evaluating its behavior. Some examples could be:

Study design choices: Explore how different design choices impact the quality of the reconstructions made by your AE. For example, you can test

having different dimensionality of the latent representations ($l = E(x)$ in Figure 1), all the way from only two dimensions up to hundreds of dimensions. You can also explore other design choices (convolutional layers, transposed convolution vs. bilinear upsampling, etc.) or test to add a noise layer after the input of your model (you will then have a denoising AE). You can measure the performance in terms of, e.g., MSE or PSNR comparing the input and output of the AE model.

Classification of AE representations: With your trained AE, you can extract the latent representations ($l = E(x)$ in Figure 1) for a dataset, and keep the class labels from the original dataset (you will need a dataset with class labels). You can then use these as input to train a classifier. In this way, you will learn how well the representation l works for separating classes. You can compare training just a single layer network to training with a deeper model, investigating how easy it is to separate the representations.

Pre-training: You can train the AE on some dataset where you also have images with class labels. If you train the AE unsupervised, you can then use the encoder to specify a classifier. For example, you can add one final dense layer with softmax activation for making classification of your images. You can then compare training this classifier from scratch, to training it by initializing the layers with the weights you trained unsupervised.

The most tricky part in this task is to set the weights of your classifier. You can have a look in the code from Lab 1-2 (the function `get_weights()`) to see how weights can be extracted from your AE. The extracted weights of the encoder can then be assigned to your classifier (if it has the same design as the encoder). You can assign weights to a layer in Keras using the function `set_weights()`. This will be something similar to:

```
W = ae.layers[i].get_weights()
classifier.layers[i].set_weights(W)
```

Here, `ae` is the Keras AE model and `classifier` is your Keras classifier model. You do this assignment for all layers i that are part of the encoder in your AE.

What is the difference in performance training the classifier from scratch as compared to training with weights initialized with the AE? You can explore doing the training of the classifier on only a subset of the training data, in which case it could be easier to see an improvement by using the AE pre-training.

Visualization of AE representations: You can also study the properties of the latent representations ($l = E(x)$ in Figure 1), e.g. using some method of visualization. For example, if you set the dimensionality of your latent representation to 2, you can plot the dataset in the latent representation. If you have more dimensions in your latent representation, you can perform dimensionality reduction to be able to show them in a 2D plot. Methods for this are, e.g., UMAP[1] or t-SNE[2]. If you plot the representations in 2D, you can color them with different colors for different class labels, to see if class information is separated in the latent representation.

4.2 Proposal 2 – Image-to-image CNNs

In this proposal, you will focus on implementing an image-to-image CNN. Here, the output of the model should be an image. You can use such model to perform different image operations, such as denoising, colorization, segmentation, in-painting, etc.

An image CNN can be formulated in its most simple form by applying a number of convolutional layers, with no down-sampling. However, to increase the receptive field, you can add pooling layers and upsampling layers, similar to how an autoencoder is formulated. For the implementation of an AE model, you can have a look at Proposal 1 in Section 4.1. However, you can formulate the model without fully-connected/dense layers. That is, you only use convolutional layers in combination with pooling and upsampling layers, as depicted in Figure 2 (left). In this way, you will have a fully-convolutional network that can be applied to images of different resolutions.

If you want to dive a bit deeper, you can apply skip-connections between the encoder and de-

[1] https://umap-learn.readthedocs.io/en/latest/

[2] https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

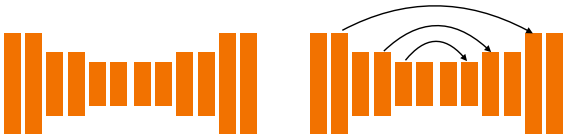

Figure 2: Fully-convolutional AE (left). With added skip-connections (right), specifying a network similar to the famous U-Net.

coder parts of the network, as illustrated in Figure 2 (right). The skip-connections can be implemented either by performing an addition, or a concatenation followed by a convolutional layer. The relevant Keras layers for doing this are:

```
keras.layers.Concatenate(...)
keras.layers.Add(...)
```

You need to make sure that the dimensionality of the added layers are the same (both resolution of channels/features and the number of channels in the convolutional layers). If you use concatenation of layers, the resolution needs to be the same but you could have different number of channels. For example, if a layer output in the encoder is $M \times N \times C_e$ and a layer output in the decoder is $M \times N \times C_d$, you can concatenate these to get an output that is $M \times N \times (C_e + C_d)$. After the concatenation, you can apply a convolutional layer with filter size 1×1 to combine the channels into size $M \times N \times C_d$.

When you have implemented the image-to-image network, you can use this for different tasks. Select a task that you find interesting, for example:

Colorization: Select a dataset with color images, and convert these to grayscale images. Train your model with the grayscale images as input and with an output of your network that is 3 channels (RGB). Formulate a loss function by comparing the output with the original RGB images. The network will then try to add colors to your grayscale images. You should not expect that this will work very well for a complex dataset, but there will be some improvements in terms of added color information.

Segmentation: You can find a dataset that provides segmentation masks for each image. Train your model to output one channel for each segmentation label, for example 2 channels if it is a binary segmentation task. The loss function could be formulated, e.g., using sparse categorical crossentropy. You can find examples of how to do this, for example in the Keras documentation[3].

Denoising: Add some noise to your input images (for example with a noise layer in Keras). Train the model to output denoised images.

In-painting: Remove some small part of the input images, for example by setting the pixels in the center region of images to 0. Train the model to output the full images, for example using the L2 loss between model output and ground truth images. You should not expect that this will work very well for a complex dataset (you would probably need to extend to a generative model, e.g. including an adversarial loss, to have high quality of the in-painted regions), but there will be some improvements in terms of added information in the missing image regions.

4.3 Proposal 3 – Generative deep learning

In this proposal, you will focus on generative models. It is a challenging task to implement a generative model, such as a GAN, from scratch. If you are up for a challenge, you can give this a go, and it is ok if you are not able to get the model to generate high quality data. Otherwise, you can find an already implemented generative model and train this on some data of choice.

A generative model could be time-consuming to train if you have high-resolution images. To facilitate the work, we suggest that you look at some simpler dataset, such as MNIST.

Some suggestions for directions to explore in this proposal:

Noise-to-image GANs: Find an implementation of a noise-to-image GAN. For example, a DCGAN could be a reasonable first model to test. This is relatively small, which makes it fast to train. There are many implementations of DCGAN out there, and one possible resource could be through

[3] https://keras.io/examples/vision/oxford_pets_image_segmentation

the Keras documentation[4]. You can also look at more state-of-the-art models, such as StyleGAN-2[5]. However, the problem with the Tensorflow implementation of StyleGAN-2 is that it requires Tensorflow 1, while the current version is Tensorflow 2. You can also install PyTorch and use a more recent implementation of StyleGAN-2[6], or look at the Keras documentation for how StyleGAN can be implemented[7].

When you have trained the GAN, you can evaluate the results in different ways. You can provide some examples of generated images and discuss the quality and properties compared to the real images. You can also evaluate the performance by means of some metric, such as the Fréchet inception distance (FID)[8]. If you want to explore further, you can test doing interpolations between input vectors to the GAN. That is, if you specify two inputs to your generator, z_1 and z_2, you can generate the interpolated inputs $z = \alpha z_1 + (1-\alpha)z_2$ for different values of α in the range $[0, 1]$. Run the different z through the generator and analyze the resulting images.

Image-to-image GANs: Find an implementation of an image-to-image GAN, such as pix2pix[9] (for paired data) or CycleGAN[10] (for unpaired data). Train the model on some dataset of choice. Remember that for pix2pix you need a reference image for each input image, while for CycleGAN you only need two different domains of images without explicit pairing between individual images. You can, for example, train pix2pix to do colorization or in-painting, as described in Section 4.2. Or you can train CycleGAN to transform between apples and oranges[11].

When you have trained the GAN, you can evaluate the results in different ways. You can provide some examples of generated images and discuss the quality and properties compared to the real images. You can also explore how the results look like if you apply some augmentations to your input images.

Implementing a GAN: If you want a challenge, you can try implementing a GAN in Keras. To make this more manageable, you can use a simple 2D dataset, such as the ones from Lab 1. You can find many resources online for aiding your development, for example looking how a GAN is implemented in the Keras documentation[12], or looking at the code provided with the Deep Generative Modeling book[13].

Diffusion models: An alternative to GANs is to test diffusion models. You can either test an existing implementation, or build a diffusion model in Keras. For testing an existing implementation, we recommend the latent diffusion model (LDM)[14], as this is efficient to run. However, the model is implemented in PyTorch, so you would need to install this. Alternatively, you can try to build a diffusion model in Keras[15], or look at the code provided with the Deep Generative Modeling book[13].

4.4 Task of choice

If you are more interested in looking at some other task, you are welcome to propose another direction for Lab 3. For example, you could look into self-supervised learning through contrastive methods, explore som explainable ML implementation, or test some neural radiance field (NeRF) implementation. You can talk to the teaching staff if you want to bounce some ideas and ensure that your project is of reasonable difficulty.

5 Conclusion

In this lab, we have looked at some more advanced applications of deep learning, mainly focusing on images. With the knowledge you have gained so far, you should hopefully have some practical experience that makes it possible to go beyond the simpler tasks and be able to use some of the many available implementations that are out there.

[4] https://keras.io/examples/generative/dcgan_overriding_train_step/
[5] https://github.com/NVlabs/stylegan2~
[6] https://nvlabs.github.io/stylegan2/versions.html
[7] https://keras.io/examples/generative/stylegan/
[8] https://github.com/jleinonen/keras-fid
[9] https://modelzoo.co/model/pix2pix-keras
[10] https://keras.io/examples/generative/cyclegan/
[11] https://www.kaggle.com/datasets/balraj98/apple2orange-dataset
[12] https://keras.io/examples/generative/dcgan_overriding_train_step/
[13] https://github.com/jmtomczak/intro_dgm
[14] https://github.com/CompVis/latent-diffusion
[15] https://keras.io/examples/generative/ddpm/