AI in Medicine I
WiSe 2022/23

Technical University Munich
Department of Informatics

I31 - AI in Medicine and Healthcare
I32 - Computational Imaging in AI and Medicine

Practical exercise 1 — 28. Nov. 2023

Transfer Learning

Submission deadline: 12. Dec. 2023, 23:59

Please submit your solutions (via Moodle).
The corresponding tutorial session is

30. Nov 2023, 16:15-17:45 in lecture hall 5901.EG.051

For questions regarding this exercise sheet, please contact: `paul.hager@tum.de` or `oezguen.turgut@tum.de`
For general questions, please contact: `course.aim-lab@med.tum.de`

Generating high quality labels for medical datasets is an expensive and time consuming task. Often we are faced with datasets that have scarce labels or none at all and must rely on self-supervised pre-training methods to increase our performance. The Jupyter Notebook provided contains some preliminary code you can use and some function prototypes that you are expected to fill in. The deliverables for this submission are an archive containing the code provided completed as well as a short report explaining your strategies and choices for each task in this practical.

1. (70%) Task 1 : **Lowdata Regime and Autoencoders**

   This week we will be working with chest x-ray datasets from MedMNIST `https://medmnist.com/`. First, we will take a look at how our network trains with very little labels, in this case 300.

   (a) (20%) 1a: **Lowdata Performance**

   Train a simple model on ChestMNIST using the code provided.

   1. The original data is multi-label. We are interested in only doing binary classification (healthy vs diseased). Fill in the train and test code to convert the data to binary.

   2. Implement the calculation of accuracy in the test function. Is accuracy appropriate here? Examine the data distribution to support your argument.

   3. What would be a better way to select a model than a fixed number of steps that would allow us to set a very high number of epochs and not overfit too much? How would we select the best model using such a "smart train" function if we are still worried about overshooting our optimum?

   (b) (20%) 1b: **Examining the Latent Space**

   Often it is useful to take a look at the latent space of our embeddings to understand how well our model separates our classes. This is especially interesting when we don't have labels and still find clusters.

   1. Adjust the Net class to return embeddings when desired.

   2. Apply PCA and t-SNE to your embeddings. Feel free to use libraries.

   (c) (30%) 1c: **Autoencoder Performance**

   When we have plenty of unlabeled data, we can use self-supervised techniques like auto-encoders to learn general features which we can then fine-tune on top of with our labels to improve our performance.

1. Write an autoencoder class that is compatible with our first model. Include dropout in the autoencoder.

2. Write a function to plot input images and their autoencoder reconstructions. Does your autoencoder work?

3. Transfer the weights from the autoencoder and finetune. Comment on the results.

2. (30%) Task 2 : **Transfer Learning and Freezing Strategies**

   PneumoniaMNIST does not have a lot of samples. Lets try improving our performance with transfer learning. First, train a model on ChestMNIST using all data using the provided code.

   (a) (10%) 2a: **Freezing Networks**

   Transfer the weights of the chest encoder to the pneumonia encoder and then freeze all layers except for the fully connected head.

   Examine the latent space.

   This is often done in the literature. Why is that? What are we testing here?

   (b) (10%) 2b: **Trainable Networks**

   Now do the same but leave the model trainable.

   Examine the latent space again.

   What worked better? Freezing or leaving the weights trainable? Why?

   (c) (10%) 2c: **Latent Space Musings**

   You've now examined the latent space using PCA and t-SNE on three seperate occasions. Compare the situations and discuss your thoughts. What do you see and why do you see this?

3. (20%) **BONUS**

   Sometimes unfreezing all layers after transferring weights can destroy the encoder because of the uninitialized head. How could you deal with this? Implement your solution.