10907 Pattern Recognition

Lecturers
Prof. Dr. Ivan Dokmanić ⟨ivan.dokmanic@unibas.ch⟩

Tutors
Felicitas Haag ⟨felicitas.haag@unibas.ch⟩
Alexandra Spitzer ⟨alexandra.spitzer@unibas.ch⟩
Cheng Shi ⟨cheng.shi@unibas.ch⟩
Vinith Kishore ⟨vinith.kishore@unibas.ch⟩

Problem set 3

Important note: For Exercise 9, we will run a leaderboard on Gradescope. The winner will receive a 4% bonus on the final exam. Students ranking from 2nd to 10th place will each receive a 3% bonus. The deadline for this task is 19.12.2025, 23:59 (end of the semester). More details are given in Exercise 9.

Math

Exercise 1 (FIR Wiener filter – ⋆)**.**

In class we derived a denoising Wiener filter which acts directly in the frequency domain. That is conceptually simple and the obtained expression is pleasingly interpretable. On the other hand, to apply this Wiener filter we always need to compute forward and inverse Fourier transforms. That can be a problem on constrained hardware in embedded devices and cameras. Even if it is not a problem, we cannot really control the filter's spatial support (the "size" of the kernel) and we should zero pad to avoid issues with circular convolution when doing the inverse transform.

It is thus often desirable to compute a convolutional Wiener filter which has an impulse response (a kernel) of fixed length. This means that we have to further restrict the class of estimators g discussed in class. We will work with 1D audio signals but the ideas extend to images.

Formally, let $\mathbf{y}$ be a noisy version of the audio signal $\mathbf{x}$ of length N, corrupted with i.i.d. zero-mean Gaussian noise of per-sample variance σ^2. In other words, $y[n] = x[n] + w[n]$ for zero-mean i.i.d. Gaussian noise $w[n]$ with per-sample variance σ^2. To compute the linear MMSE estimator we were solving

$$\min_{\mathbf{H}} \ \mathbb{E}\ \|\mathbf{H}(\mathbf{y} - \mu_Y) + \mu_X - \mathbf{x}\|^2.$$

The expectation averages over the joint distribution of $\mathbf{x}$ and $\mathbf{y}$ (or equivalently over $\mathbf{x}$ and the noise). We then restricted $\mathbf{H}$ to be a convolution and we obtained the Wiener filter in the frequency domain.[1] Now we further restrict $\mathbf{H}$ to be a convolution with a filter $\mathbf{h}$ of fixed size $L + 1$, i.e. $\mathbf{Hy} = \mathbf{y} * \mathbf{h}$. For simplicity we'll assume that $\mu_Y = \mu_X = 0$ but that's easy to revert. (It's definitely true for audio recorded on common hardware which does not let the DC component pass.) The filter size is often much smaller than the signal length, $L \ll N$. We will use the explicit convolution notation and denote the filter

$$\mathbf{h} = \begin{bmatrix} h[0] \\ h[1] \\ \vdots \\ h[L] \end{bmatrix}.$$

The filtered signal is then

$$\widehat{x}[n] = (y * h)[n] = \sum_{m=0}^{L} h[m]y[n-m].$$

[1] If you are reading this statement before Friday, then it is anti-causal.

We want to find $\mathbf{h}$ which minimizes the MSE *for each* n (optimality of this follows from certain stationarity assumptions),

$$\mathbb{E}\left[(\widehat{x}[n]-x[n])^2\right]=\mathbb{E}\left[\left(x[n]-\sum_{m=0}^{L}h[m]y[n-m]\right)^2\right].$$

1. Starting from the above expression, show that the coefficients of the optimal filter satisfy

$$\mathbb{E}\Big[\,(x[n]-\widehat{x}[n])\,y[n-\ell]\Big]=\mathbb{E}\left[\left(x[n]-\sum_{m=0}^{L}h[m]y[n-m]\right)y[n-\ell]\right]=0,\quad \ell=0,1,\ldots,L.$$

   Hint: differentiate the expected error with respect to $h[\ell]$.

2. For each ℓ between 0 and L, the above expression yields a linear equation for the filter coefficients. We thus obtain $L+1$ linear equations in $L+1$ unknowns which can be succinctly written as

$$\mathbf{R}\mathbf{h}=\mathbf{r}.$$

   Find the $(L+1)\times(L+1)$ matrix $\mathbf{R}$ and the vector $\mathbf{r}$. *Hint: you should see some covariances (expectations of products) pop up.*

Exercise 2 (Discrete Fourier Transform – ⋆⋆)**.**

Recall that the discrete Fourier transform (DFT) of a vector $\mathbf{x}=[x[0],x[1],\ldots,x[N-1]]^T$ of length N is defined as

$$X[k]=(F\mathbf{x})[k]=\sum_{n=0}^{N-1}x[n]e^{-j2\pi kn/N},\quad k\in\{0,1,\ldots,N-1\}.$$

The inverse DFT of a vector $\boldsymbol{X}=[X[0],X[1],\ldots,X[N-1]]^T$ as

$$x[n]=(F^{-1}\boldsymbol{X})[n]=\frac{1}{N}\sum_{k=0}^{N-1}X[k]e^{j2\pi kn/N},\quad n\in\{0,1,\ldots,N-1\}.$$

1. We have seen in class that any linear operator in finite dimensions can be represented by a matrix. Fourier transform is a linear operator. Determine the matrices representing F and F^{-1} such that $\boldsymbol{X}=F\mathbf{x}$ and $\mathbf{x}=F^{-1}\boldsymbol{X}$.

2. The inverse DFT is, not surprisingly, the inverse of the forward DFT operator. Show that $F^{-1}(F\mathbf{x})=\mathbf{x}$ by directly multiplying the matrices. *Hint: use the formula for the sum of a geometric series to show that* $F^{-1}F=I$.

3. Show that $F^{-1}=\dfrac{1}{N}F^*$ where F^* is obtained from F by transposing and then applying complex conjugation to all entries. (Btw, do we need the transpose in this case?) This implies that up to a rescaling, the DFT matrix is unitary meaning that its rows (and columns!) are orthonormal. Had we defined a slightly different transform $\tilde{F}$ where factors 1 and $\frac{1}{N}$ in the forward and inverse transform are both replaced by $1/\sqrt{N}$ then we would have that $\tilde{F}^{-1}=\tilde{F}^*$, which is to say that $\tilde{F}$ is a unitary matrix. Check that this is indeed the case.

4. Show that

$$\sum_{k=0}^{N-1}|X[k]|^2=N\sum_{n=0}^{N-1}|x[n]|^2.$$

   This fact, that the ℓ_2-norm can be computed in either the spatial domain or the frequency domain, is called Parseval's theorem.

Exercise 3 (DFT and circular convolution - ⋆⋆)**.**

Let $\mathbf{x}$ be a vector of length N. We define a circular shift of $\mathbf{x}$ by an amount n_0 as

$$(\text{circshift}_{n_0}(\mathbf{x}))[n] = x[(n-n_0)_N] = \begin{cases} x[n-n_0] \text{ if } n \geq n_0 \\ x[N-(n_0-n)] \text{ if } n < n_0 \end{cases} .$$

Let $\boldsymbol{X}$ be the DFT of $\mathbf{x}$.

1. Show that when applying the DFT to a circularly-shifted signal, the new DFT coefficients are just the DFT coefficients of the original signal modulated by a complex exponential:

$$(F(\text{circshift}_{n_0}(\mathbf{x})))[k] = X[k]e^{-j2\pi kn_0/N}.$$

2. Circular convolution of two vectors $\mathbf{w}$ and $\mathbf{x}$ of length N is defined as

$$(\mathbf{w} \circledast \mathbf{x})[n] = \sum_{\ell=0}^{N-1} x[\ell]w[(n-\ell)_N].$$

Let $\boldsymbol{W}$ be the DFT of $\mathbf{w}$. Then show

$$(F(\mathbf{w} \circledast \mathbf{x}))[k] = W[k]X[k].$$

This relation allows us to efficiently compute circular convolution of two signals using their DFTs.

Exercise 4 (Filter design - ⋆⋆⋆)**.**

Work with the $N = 16$ point DFT and circular convolution modulo N. Let $h[n]$ be a real sequence with support contained in $\{0, \ldots, L\}$ for some $L \leq 15$. Denote its DFT by

$$H[k] \;=\; \sum_{n=0}^{15} h[n]\, e^{-j2\pi kn/16}, \qquad k \in \{0, 1, \ldots, 15\}.$$

We say that h has *shortest support* if L is as small as possible among all sequences satisfying the constraints below.

1. Design a nontrivial real sequence $h[n]$ whose DFT satisfies

$$H[3] \;=\; 0, \qquad H[10] \;=\; 0.$$

Interpret "shortest support" precisely as: find the minimal L for which there exists h with $h[n] = 0$ for all $n > L$ and the above DFT constraints hold.

2. Design three different nonzero signals $x^{(i)}[n]$ of length 16 such that the circular convolution

$$(h \circledast x^{(i)})[n] \;=\; \sum_{\ell=0}^{15} h[\ell]\; x^{(i)}[(n-\ell)_{16}]$$

is identically zero for each $i = 1, 2, 3$. Express your construction in the frequency domain using the identity

$$\mathcal{F}\{h \circledast x\}[k] \;=\; H[k]\; X[k],$$

and explain why your choices force $H[k]X^{(i)}[k] = 0$ for all k.

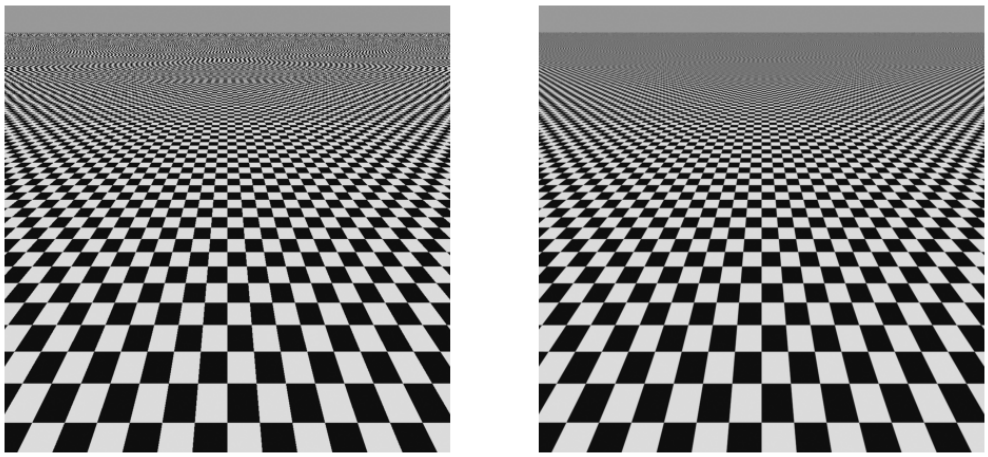
Exercise 5 (Fun with aliasing - $\star\star\star$)**.**

Answer the following questions:

- Let $x_1[n] = \cos(2\pi/Nk_1n)$, $x_2[n] = \cos(2\pi/Nk_2n)$ be two real sinusoids of length $N = 32$, with $k_1 = 5$ and $k_2 = 13$. Let also $y_1[n] = x_1[4n]$ and $y_2[n] = x_2[4n]$ be their downsampled versions of length $N/4 = 8$. Plot x_1, x_2, y_1, y_2 and interpret the result.
- A length-16 real sinusoid is known to be $x[n] = \cos(2\pi(k_0/16)n)$ with $k_0 \in \{1, 3, 5, 7\}$. You only get to see every fourth sample, $y[n] = x[4n]$ for $n = 0, 1, 2, 3$. If $y = (1, 0, -1, 0)$ what are the possible values of k_0?
- Let $x[n]$ be a discrete signal of even length N, with the DFT $X[k]$. Let $y[n] = x[2n]$ be a downsampled version of x of length $N/2$. Write its DFT $Y[k]$ in terms of $X[k]$.
- Let now $x[m, n]$ be an image of size $N \times N$ where N is even, with the DFT $X[u, v]$, and $y[m, n] = x[2m, 2n]$ a downsampling by 2. Write $Y[u, v]$ in terms of $X[u, v]$.
- What is the significance of these results for image downsampling in convolutional neural nets?

Exercise 6 (More fun with aliasing - $\star$)**.**

The figure below shows two regular checkerboard patterns in perspective. Can you explain what is going on? Why are there strong snaking psychedelic patterns on the left (and a little bit on the right)? *Hint: on the right, the camera lens blurs more than on the left. Use what we learned about sampling and aliasing.*



Coding

Large deep learning models benefit from GPU acceleration. You can check whether you are using a GPU by looking at the output of `torch.cuda.is_available()` (on your own machine or on sciCORE). For the problem sizes here, training on a CPU will also be fine. We will use the MNIST dataset in all three exercises. You can use the function `data_generator` to generate and manipulate the dataset (the first call may take longer since it will download the images). Do not modify any function in the file `helper.py`. This will help you discover potential problems with your code that could occur during autograding.

Exercise 7 (Denoising - $\star$)**.**

Denoising aims to recovering an unknown, clean image u from its noisy version

$$y = u + \eta,$$

where we will take η to be additive white Gaussian noise. This means that each pixel (i, j) of the original image is degraded by adding an independent random variable $\eta[i, j] \sim \mathcal{N}(0, \sigma^2)$. In this exercise, we assume that σ is fixed and equal to 0.5.
In the lecture you will learn how to use filtering for denoising. In this exercise, we are interested in using deep neural networks for denoising. *Remarkably, denoising with convolutional neural networks is the key component behind the powerful generative diffusion models. We will learn about it in a couple of weeks.*
We will first focus on the very simple convolutional neural network (CNN) defined in the deep learning bootcamp. The architecture is defined in `CNN_simple` in the file `helper.py`.
We also provide the function `add_noise.py` that adds Gaussian random noise to a batch of images. Details of the input dimension and arguments of the function can be found in the file `helper.py`. This function can be used to generate noisy images to train a neural network.

Your task: train the neural network implemented in the function `CNN_simple` to denoise images from the MNIST dataset.
Your trained network will be evaluated on another part of the MNIST dataset with additive Gaussian noise with $\sigma = 0.5$. We will quantify the quality of the estimation by using the Signal to Noise Ratio (SNR), which is a common metric to assess the quality of reconstructed images compared to the true ones. You can check the SNR obtained with your trained network by running the file `denoising.py` on the MNIST test dataset. In this script, we compute the SNR between the denoised images and the original images. The higher the SNR, the closer the estimate is to the clean image. The autograder grades this exercise according to the SNR obtained with your trained network.

Some rules to follow for successful autograding:

- Assuming that your torch architecture is called `net` (obtained from `net=CNN_simple(image_size)`), you must save your model with the following command:

  `torch.save(net.state_dict(), "denoising.pt")`,

  and provide the file `denoising.pt` in your submission.
- Do not modify the neural network architecture in the file `helper.py`.

Once the model is saved, you can run the script `denoising.py` to check the results of your network.
Hints:

- Try different losses: `https://pytorch.org/docs/stable/nn.html#loss-functions`. The most common losses for this application being `MSELoss` and `L1Loss`.

- Make sure to use the full training dataset and to train for enough epochs.
- Adjust the learning rate.

Exercise 8 (Deblurring - ⋆⋆)**.**

Deblurring estimates a sharp image u from its blurry observation, which we model as convolution with a blur filter $h \in \mathbb{R}^{N\times N}$:

$$y = h * u + \eta,$$

where η is an additive white Gaussian noise (as in the previous exercise). In this part, we consider smaller noise with $\sigma = 0.01$.
The blur kernel h is given by a Gaussian convolution filter with standard deviation $\sigma_{blur} = 1.5$ and normalized to sum to one. The blur kernel can be obtained using the function `Gaussian_blur` in the file `helper.py`. The convolution can be computed in Pytorch using the function `convolution_fft_torch` from `helper.py`.
Your task: train a neural network to perform deblurring. In this exercise, you can design any neural network architecture using Pytorch. As we have discussed during the deep learning bootcamp and the lectures, neural networks based on convolutions are usually preferred in image processing applications. In particular, autoencoder and Unet-like structures are particularly efficient. But feel free to go to town and use vision transformers or state-space models or ... if you like!
Just note that we will evaluate your trained network on another part of the MNIST dataset while applying a Gaussian blur with $\sigma_{blur} = 1.5$ and additive Gaussian noise with $\sigma = 0.01$. The autograder grades this exercise according to the SNR similarly to the previous exercise.

Some rules to follow for successful grading:

- Assuming that your torch architecture is called `net`, you must save your model with the following command:

```
model_scripted = torch.jit.script(net.cpu())
model_scripted.save("deblurring.pt")
```

  and provide the file `deblurring.pt` in your submission.
- Once saved, the file `deblurring.pt` must **not exceed** 15mb.

Once the model is saved, you can run the script `deblurring.py` to check the results of your network. If the script runs (without modification) and you get a good SNR improvement, then you are guaranteed to get most of the autograder's points.

Hint:

- You can use the function `Upsample` from Pytorch to upsample an image (useful for autoencoder and Unet like architectures).
- Adding two upsampling layers at the end of the CNN defined in the previous exercise should already provide very accurate results.

Exercise 9 (Semi(blind) deblurring (top 10 get bonus for final exam!) - ⋆⋆⋆)**.**

In this exercise, we observe a blurry-noisy image y such that:

$$y = h * u + \eta,$$

where η is an additive white Gaussian noise. However, this time, we assume that the Gaussian filter h is not fixed but is defined with standard deviation uniformly chosen at random in the interval $[0.5, 2.5]$.
Your task: train a neural network (any architecture) that can deal with such uncertainty about the blur operator.

Some rules to follow for successful grading:

- Assuming that your torch architecture is called `net`, you must save your model with the following command:

```
model_scripted = torch.jit.script(net.cpu())
model_scripted.save("deblurring_family.pt")
```

  and provide the file `deblurring_family.pt` in your submission.

- Once saved, the file `deblurring_family.pt` must **not exceed** 15 MB.
- Name your submission with your real name.

Once the model is saved, you can run the script `deblurring_family.py` to check the results of your network.

For this exercise, we will run a leaderboard on Gradescope. The student achieving the highest SNR (the higher, the better) will receive a 4% bonus on the final exam. Students ranking from 2nd to 10th place will each receive a 3% bonus. The deadline for this task is 19.12.2025, 23:59 (end of the semester)