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 4

Math

Exercise 1 (Conditional expectations and a new kind of orthogonality - $\star$)**.** Let (X, Y) be two real-valued jointly distributed random variables and define their "$\mathbb{P}$ inner product" as

$$\langle \mathsf{X}, \mathsf{Y} \rangle_{\mathbb{P}} = \mathbb{E}(\mathsf{XY}).$$

Check quickly that this is indeed an inner product. The corresponding norm is given as

$$\|\mathsf{X}\|_{\mathbb{P}}^2 = \langle \mathsf{X}, \mathsf{X} \rangle_{\mathbb{P}} = \mathbb{E}(\mathsf{X}^2).$$

Let now

$$\widehat{\mathsf{X}} = \arg\min \mathbb{E}[\mathsf{X} \mid 04_p set_P R25Y]$$

be the MMSE estimator of X given Y. Show that the estimator error $\widehat{\mathsf{X}} - \mathsf{X}$ is orthogonal to Y in the geometry induced by $\langle \cdot, \cdot \rangle_{\mathbb{P}}$, that is to say

$$\left\langle \widehat{\mathsf{X}} - \mathsf{X}, \mathsf{Y} \right\rangle_{\mathbb{P}} = 0$$

Try to interpret this in words. How can we think of the conditional expectation?

Hint: First show that

$$\mathbb{E}[\mathbb{E}[\mathsf{XY} \mid \mathsf{Y}]] = \mathbb{E}[\mathsf{Y}\mathbb{E}[\mathsf{X} \mid \mathsf{Y}]] \quad \textit{and} \quad \mathbb{E}[\mathbb{E}[\mathsf{X}|\mathsf{Y}]] = \mathbb{E}[\mathsf{X}].$$

You can assume that all the needed joint and conditional probability density functions exist.

For real-valued random variables X, Y, Z with $\mathbb{E}[X^2], \mathbb{E}[Y^2], \mathbb{E}[Z^2] < \infty$ define

$$\langle X, Y \rangle_{\mathbb{P}} := \mathbb{E}[XY].$$

Then

- symmetry: $\langle X, Y \rangle_{\mathbb{P}} = \mathbb{E}[XY] = \mathbb{E}[YX] = \langle Y, X \rangle_{\mathbb{P}}$;
- linearity in the first argument:
$$\langle aX + bY, Z \rangle_{\mathbb{P}} = \mathbb{E}[(aX + bY)Z] = a\mathbb{E}[XZ] + b\mathbb{E}[YZ] = a\langle X, Z \rangle_{\mathbb{P}} + b\langle Y, Z \rangle_{\mathbb{P}};$$
- positive definiteness:
$$\langle X, X \rangle_{\mathbb{P}} = \mathbb{E}[X^2] \geq 0,$$
and $\mathbb{E}[X^2] = 0$ implies $X = 0$.

This verifies that $\langle \cdot, \cdot \rangle_{\mathbb{P}}$ is indeed an inner product.

By the law of total expectation (tower property), $\mathbb{E}[\mathbb{E}[X \mid Y]] = \mathbb{E}[X]$. (If a density $f_{X,Y}$ exists, this can be checked by writing out the corresponding integrals.) To show that, $\mathbb{E}[\mathbb{E}[XY \mid Y]] = \mathbb{E}[Y\,\mathbb{E}[X \mid Y]]$, we first compute the conditional expectation:

$$\mathbb{E}[XY \mid Y] = Y\,\mathbb{E}[X \mid Y].$$

Indeed, for $Y = y$,

$$\mathbb{E}[XY \mid Y = y] = \int xy f_{X|Y}(x \mid y)\, dx = y \int x f_{X|Y}(x \mid y)\, dx = y\, \mathbb{E}[X \mid Y = y].$$

Thus, as random variables,

$$\mathbb{E}[XY \mid Y] = Y\, \mathbb{E}[X \mid Y].$$

Taking expectations on both sides gives

$$\mathbb{E}[\mathbb{E}[XY \mid Y]] = \mathbb{E}\big[Y\, \mathbb{E}[X \mid Y]\big].$$

The MMSE estimator of X given Y is $\hat{X} = \mathbb{E}[X \mid Y]$. Combining the equality above with the law of total expectation we get,

$$\mathbb{E}[XY] = \mathbb{E}[\mathbb{E}[XY \mid Y]] = \mathbb{E}\big[Y\, \mathbb{E}[X \mid Y]\big] = \mathbb{E}\big[Y\hat{X}\big].$$

To show that the error $\hat{X} - X$ is orthogonal to Y in $\langle \cdot, \cdot \rangle_{\mathbb{P}}$, i.e. $\big\langle \hat{X} - X, Y \big\rangle_{\mathbb{P}}$, compute

$$\begin{aligned}
\big\langle \hat{X} - X, Y \big\rangle_{\mathbb{P}} &= \mathbb{E}\big[(\hat{X} - X)Y\big] \\
&= \mathbb{E}[\hat{X}Y] - \mathbb{E}[XY] \\
&= \mathbb{E}\big[Y\, \mathbb{E}[X \mid Y]\big] - \mathbb{E}[XY] \\
&= \mathbb{E}[\mathbb{E}[XY \mid Y]] - \mathbb{E}[XY] \\
&= \mathbb{E}[XY] - \mathbb{E}[XY] \\
&= 0.
\end{aligned}$$

Exercise 2 (Cascading convolutions - ⋆⋆)**.** Let $\boldsymbol{A}$ and $\boldsymbol{B}$ be two circulant matrices of size $N \times N$. Show in two different ways that $\boldsymbol{AB}$ and $\boldsymbol{BA}$ are also circulant:

- Using direct calculations for matrix multiplication;
- Using the convolution–multiplication rule for discrete Fourier transforms ($\equiv$ the fact that circular convolutions are diagonalized by the DFT).

A matrix $A \in \mathbb{C}^{N \times N}$ is *circulant* if its entries satisfy

$$A_{ij} = a_{(i-j) \bmod N},$$

for some vector $a = (a_0, \dots, a_{N-1})^T$, called the *generating vector*. Similarly, B is circulant with generating vector b. Then

$$A = \operatorname{circ}(a) = \begin{pmatrix}
a_0 & a_{N-1} & a_{N-2} & \cdots & a_2 & a_1 \\
a_1 & a_0 & a_{N-1} & \cdots & a_3 & a_2 \\
a_2 & a_1 & a_0 & \cdots & a_4 & a_3 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
a_{N-2} & a_{N-3} & a_{N-4} & \cdots & a_0 & a_{N-1} \\
a_{N-1} & a_{N-2} & a_{N-3} & \cdots & a_1 & a_0
\end{pmatrix}.$$

Direct calculation Let $C = AB$. Then

$$C_{ij} = (AB)_{ij} = \sum_{k=0}^{N-1} A_{ik} B_{kj} = \sum_{k=0}^{N-1} a_{(i-k) \bmod N}\, b_{(k-j) \bmod N}.$$

Perform the change of variables using the congruence[1]

$$\ell \equiv i - k \pmod{N}, \qquad 0 \le \ell \le N - 1.$$

[1]

For fixed i, the map

$$k \longmapsto (i-k) \bmod N$$

is a permutation of $\{0,\dots,N-1\}$. Hence, as k runs through $\{0,\dots,N-1\}$, so does ℓ, exactly once each, and we may replace $\sum_{k=0}^{N-1}$ by $\sum_{\ell=0}^{N-1}$. From $\ell \equiv i-k \pmod N$ we obtain

$$k \equiv i - \ell \pmod N,$$

and therefore

$$(k-j) \bmod N \equiv (i-\ell-j) \bmod N = (i-j-\ell) \bmod N.$$

Substituting this into the sum yields

$$C_{ij} = \sum_{\ell=0}^{N-1} a_\ell \, b_{(i-j-\ell) \bmod N}.$$

If we now define $m = (i-j) \bmod N$ and

$$c_m := \sum_{\ell=0}^{N-1} a_\ell \, b_{(m-\ell) \bmod N},$$

then

$$C_{ij} = c_{(i-j) \bmod N},$$

which shows that C is circulant with generating vector c. Note that c is the circular convolution... why?

Using the DFT Let F be the unitary DFT matrix:

$$F_{k\ell} = \frac{1}{\sqrt{N}} e^{-2\pi i k\ell/N}, \qquad k,\ell = 0,\dots,N-1.$$

Every circulant matrix is diagonalized by F. If A has generating vector a, then

$$A = F^{-1}\mathrm{diag}(\widehat{a})F,$$

where $\widehat{a} = Fa$ is the DFT of a. Similarly,

$$B = F^{-1}\mathrm{diag}(\widehat{b})F.$$

Compute:

$$\begin{aligned} AB &= (F^{-1}\mathrm{diag}(\widehat{a})F)(F^{-1}\mathrm{diag}(\widehat{b})F) \\ &= F^{-1}\mathrm{diag}(\widehat{a})(FF^{-1})\mathrm{diag}(\widehat{b})F \\ &= F^{-1}(\mathrm{diag}(\widehat{a})\mathrm{diag}(\widehat{b}))F. \end{aligned}$$

The product $\mathrm{diag}(\widehat{a})\mathrm{diag}(\widehat{b})$ is diagonal with diagonal entries

$$(\mathrm{diag}(\widehat{a})\mathrm{diag}(\widehat{b}))_{kk} = \widehat{a}_k \, \widehat{b}_k.$$

Thus

$$AB = F^{-1}\, \mathrm{diag}(\widehat{a} \odot \widehat{b})\, F,$$

which is exactly the diagonalization formula of a circulant matrix. Hence AB is circulant.

Exercise 3 (Filling in the steps from lecture notes - $\star\star$)**.** Let $\mathbf{w}$ be a random image of size $N \times N$ with pixels $\mathsf{w}[m,n]$. Let also

$$a_{\mathbf{ww}}[k,\ell] = \mathbb{E}\big[\mathsf{w}[m,n]\mathsf{w}[(m+k)_N,(n+\ell])_N]\big]$$

be its (periodic) autocorrelation function at shift (k,ℓ) (NB: if $\mathbf{w}$ were complex-valued we'd have to conjugate the shifted image). We implicitly assumed that the distribution of $\mathbf{w}$ is (wide-sense)[2] stationary. Show that the DFT of $a_{\mathbf{ww}}$ is its power-spectral density (PSD), defined as

$$S_{\mathbf{ww}}[u,v] = \mathbb{E}\ |\mathsf{W}[u,v]|^2$$

where $\mathbf{W}$ is the discrete Fourier transform of $\mathbf{w}$ (itself a random vector).

Show furthermore that under stationarity the different frequencies are uncorrelated; that is, that

$$\mathbb{E}\ \mathsf{W}[u,v]\overline{\mathsf{W}}[u',v'] = 0$$

when $(u,v) \neq (u',v')$, with overline denoting complex conjugation.

Let $w[m,n]$, $m,n \in \{0,\dots,N-1\}$, be a real (or complex) random image of size $N \times N$ and define its (periodic) autocorrelation function

$$a_{ww}[k,\ell] \;=\; \mathbb{E}\big[w[m,n]\,w[(m+k)_N,(n+\ell)_N]\big], \qquad k,\ell \in \{0,\dots,N-1\}.$$

Wide-sense stationarity implies that $a_{ww}[k,\ell]$ does not depend on (m,n).

We use the 2-D DFT (any consistent normalisation will do; constants only change overall scale)

$$W[u,v] = \sum_{m,n} w[m,n]\,e^{-j\frac{2\pi}{N}(um+vn)}, \qquad w[m,n] = \frac{1}{N^2}\sum_{u,v} W[u,v]\,e^{j\frac{2\pi}{N}(um+vn)}.$$

DFT of the autocorrelation equals the PSD. Compute the squared magnitude of $W[u,v]$:

$$\begin{aligned}
|W[u,v]|^2 &= W[u,v]\overline{W[u,v]} \\
&= \Big(\sum_{m,n} w[m,n]\,e^{-j\frac{2\pi}{N}(um+vn)}\Big)\Big(\sum_{p,q} w[p,q]\,e^{j\frac{2\pi}{N}(up+vq)}\Big) \\
&= \sum_{m,n}\sum_{p,q} w[m,n]\,w[p,q]\,e^{-j\frac{2\pi}{N}\big(u(m-p)+v(n-q)\big)}.
\end{aligned}$$

Introduce the shift variables[3] $k = (p-m)_N$, $\ell = (q-n)_N$, so that $p = (m+k)_N$, $q = (n+\ell)_N$. Then

$$u(m-p) + v(n-q) = -uk - v\ell,$$

hence

$$|W[u,v]|^2 = \sum_{m,n}\sum_{k,\ell} w[m,n]\,w[(m+k)_N,(n+\ell)_N]\,e^{j\frac{2\pi}{N}(uk+v\ell)}.$$

Taking expectations and using the definition of a_{ww},

$$\begin{aligned}
\mathbb{E}\big[|W[u,v]|^2\big] &= \sum_{m,n}\sum_{k,\ell} \mathbb{E}\big[w[m,n]w[(m+k)_N,(n+\ell)_N]\big]\,e^{j\frac{2\pi}{N}(uk+v\ell)} \\
&= \sum_{m,n}\sum_{k,\ell} a_{ww}[k,\ell]\,e^{j\frac{2\pi}{N}(uk+v\ell)} \\
&= N^2\sum_{k,\ell} a_{ww}[k,\ell]\,e^{j\frac{2\pi}{N}(uk+v\ell)}.
\end{aligned}$$

Up to the overall normalisation constant N^2 (which depends on the DFT convention), this is exactly the 2-D DFT of a_{ww}. Thus the DFT of the autocorrelation is the power spectral density

$$S_{ww}[u,v] \;:=\; \mathbb{E}\big[|W[u,v]|^2\big].$$

[2]In 2D WSS: (i) $\mathbb{E}[w[m,n]] = \mu$ for all m,n; (ii) $\mathbb{E}[w[m,n]\,w[p,q]] = a_{ww}[p-m,\ \ q-n]$ (mod N due to periodicity).

[3]Remember that $k = (p-m)_N \iff k \equiv p - m \mod N$.

Uncorrelatedness of different frequencies. Now consider two (possibly different) frequency indices (u, v) and (u', v'):

$$\begin{aligned}\mathbb{E}\big[W[u,v]\overline{W[u',v']}\big] &= \mathbb{E}\left[\sum_{m,n} w[m,n]e^{-j\frac{2\pi}{N}(um+vn)} \sum_{p,q} w[p,q]e^{j\frac{2\pi}{N}(u'p+v'q)}\right] \\ &= \sum_{m,n}\sum_{p,q} \mathbb{E}\big[w[m,n]w[p,q]\big]\, e^{-j\frac{2\pi}{N}\left(um+vn-u'p-v'q\right)}.\end{aligned}$$

Using wide-sense stationarity, $\mathbb{E}[w[m,n]w[p,q]] = a_{ww}[p-m, q-n]$. Again set $k = (p-m)_N$, $\ell = (q-n)_N$:

$$\begin{aligned}\mathbb{E}\big[W[u,v]\overline{W[u',v']}\big] &= \sum_{m,n}\sum_{k,\ell} a_{ww}[k,\ell]\, e^{-j\frac{2\pi}{N}\left((u-u')m+(v-v')n-u'k-v'\ell\right)} \\ &= \left(\sum_{m=0}^{N-1} e^{-j\frac{2\pi}{N}(u-u')m}\right)\left(\sum_{n=0}^{N-1} e^{-j\frac{2\pi}{N}(v-v')n}\right)\sum_{k,\ell} a_{ww}[k,\ell]\, e^{j\frac{2\pi}{N}(u'k+v'\ell)}.\end{aligned}$$

The discrete complex exponentials are orthogonal:

$$\sum_{m=0}^{N-1} e^{-j\frac{2\pi}{N}(u-u')m} = N\,\delta_{u,u'}.$$

To show this, let $r := e^{-j\frac{2\pi}{N}(u-u')}$. Then

$$\sum_{m=0}^{N-1} e^{-j\frac{2\pi}{N}(u-u')m} = \sum_{m=0}^{N-1} r^m.$$

We distinguish two cases.

Case 1: $u = u'$ Then $u - u' = 0$ and therefore $r = e^0 = 1$. Hence

$$\sum_{m=0}^{N-1} r^m = \sum_{m=0}^{N-1} 1 = N.$$

Case 2: $u \neq u'$ Then $r \neq 1$, but

$$r^N = e^{-j\frac{2\pi}{N}(u-u')N} = e^{-j2\pi(u-u')} = \cos(-2\pi(u-u')) - j\sin(2\pi(u-u')) = 1 - 0.$$

Using the geometric series formula,

$$\sum_{m=0}^{N-1} r^m = \frac{1-r^N}{1-r} = \frac{1-1}{1-r} = 0.$$

Combining the two cases, we obtain

$$\sum_{m=0}^{N-1} e^{-j\frac{2\pi}{N}(u-u')m} = N\,\delta_{u,u'} = \begin{cases} N, & u = u' \\ 0, & u \neq u' \end{cases}$$

Therefore

$$\mathbb{E}\big[W[u,v]\overline{W[u',v']}\big] = N^2\delta_{u,u'}\delta_{v,v'}\sum_{k,\ell} a_{ww}[k,\ell]\, e^{j\frac{2\pi}{N}(u'k+v'\ell)}.$$

In particular, if $(u,v) \neq (u',v')$ then at least one of the deltas is zero and

$$\mathbb{E}\big[W[u,v]\overline{W[u',v']}\big] = 0.$$

Thus, for a wide-sense stationary random image the DFT coefficients at different frequencies are uncorrelated.

Exercise 4 (Fun with CNNs - ⋆⋆⋆)**.** The following listing defines a simple U-Net mapping images to images (adapted from `https://github.com/clemkao/u-net`):

```python
import torch
import torch.nn as nn
from typing import List

class DoubleConv(nn.Module):
    """(Conv -> BN -> ReLU) x 2, 'same' spatial size for k=3"""
    def __init__(self, in_ch, out_ch, dilation=1):
        super().__init__()
        pad = dilation
        self.net = nn.Sequential(
            nn.Conv2d(in_ch, out_ch, kernel_size=3, padding=pad, dilation=dilation,
            ↪  bias=False),
            nn.BatchNorm2d(out_ch),
            nn.ReLU(inplace=True),
            nn.Conv2d(out_ch, out_ch, kernel_size=3, padding=pad, dilation=dilation,
            ↪  bias=False),
            nn.BatchNorm2d(out_ch),
            nn.ReLU(inplace=True),
        )

    def forward(self, x):
        return self.net(x)

class Down(nn.Module):
    """Downsample by 2 then DoubleConv"""
    def __init__(self, in_ch, out_ch, dilation=1):
        super().__init__()
        self.pool = nn.MaxPool2d(2)
        self.conv = DoubleConv(in_ch, out_ch, dilation=dilation)

    def forward(self, x):
        return self.conv(self.pool(x))

class Up(nn.Module):
    """Upsample by 2 then concat skip and DoubleConv"""
    def __init__(self, in_ch, out_ch):
        super().__init__()
        self.up = nn.ConvTranspose2d(in_ch, in_ch // 2, kernel_size=2, stride=2)
        self.conv = DoubleConv(in_ch, out_ch)

    def forward(self, x, skip):
        x = self.up(x)
        # pad if needed (odd sizes)
        diffY = skip.size(2) - x.size(2)
        diffX = skip.size(3) - x.size(3)
        if diffY or diffX:
            x = nn.functional.pad(x, [diffX // 2, diffX - diffX // 2,
                                      diffY // 2, diffY - diffY // 2])
        x = torch.cat([skip, x], dim=1)
        return self.conv(x)

class OutConv(nn.Module):
    def __init__(self, in_ch, out_ch):
        super().__init__()
        self.conv = nn.Conv2d(in_ch, out_ch, kernel_size=1)

    def forward(self, x):
        return self.conv(x)

class SimpleUNet(nn.Module):
    def __init__(self, in_ch=1, out_ch=1, base=32, depth=4, dilations: List[int] =
    ↪ None):
        super().__init__()
        if dilations is None:
            dilations = [1] * depth
        assert len(dilations) == depth

        ch = [base * (2 ** i) for i in range(depth)]  # e.g., 32, 64, 128, 256

        # encoder
        self.inc   = DoubleConv(in_ch, ch[0], dilation=dilations[0])
        self.downs = nn.ModuleList([
            Down(ch[i], ch[i+1], dilation=dilations[i+1]) for i in range(depth-1)
        ])

        # decoder
        self.ups = nn.ModuleList([
            Up(ch[i], ch[i-1]) for i in range(depth-1, 0, -1)
        ])

        self.outc = OutConv(ch[0], out_ch)

    def forward(self, x):
        xs = [self.inc(x)]
        for d in self.downs:
            xs.append(d(xs[-1]))
        y = xs[-1]
        for i, up in enumerate(self.ups, start=1):
            y = up(y, xs[-i-1])
        return self.outc(y)
```

- Give a formula for the total number of network parameters in terms of the four hyperparameters (`in_ch`, `out_ch`, `base`, `depth`). How much do you get for `in_ch = 3`, `out_ch = 1`, `base = 32`, `depth = 4`? Which parts dominate the parameter count?
- Compute the output receptive field R; that is, given a pixel in the output image, which pixels (how many) in the input image can it depend on?
- How would you increase the receptive field without increasing the number of parameters too much? (*Hint:* check out dilated convolutions.)
- Is `SimpleUNet` linear?
- Explain what will happen if we replace the ReLU activations with linear activations $r(x) = x$.
- We like convolutions because they are translation invariant. `SimpleUNet` is a convolutional neural net. Is it strictly translation invariant (ignoring boundary effects)?

1. We compute the number of parameters in the U-Net by evaluating each block separately and summing their contributions.

   A `DoubleConv` block contains two convolutional layers (each without bias) followed by batch-normalization layers, which together contribute $2\,\text{out\_ch}$ parameters. Thus, the total number of parameters in a `DoubleConv` block is

$$9\,\text{in\_ch}\cdot\text{out\_ch} \;+\; 9\,\text{out\_ch}^2 \;+\; 4\,\text{out\_ch}.$$

A `Down` block contains the same number of trainable parameters as a `DoubleConv` block (what about maxpool ?), and therefore has the same parameter count. A `Up` block, however, includes an additional transposed convolution, which contributes

$$2\,\text{in\_ch}^2 \;+\; \tfrac{1}{2}\,\text{in\_ch}$$

parameters in addition to the parameters of the associated `DoubleConv`.

Parameter count for the SimpleUNet: The network begins with an initial block, mapping `in_ch` channels to `base` channels, giving

$$9\,\text{in\_ch}\cdot\text{base} \;+\; 9\,\text{base}^2 \;+\; 4\,\text{base}.$$

In the downsampling path, the number of channels doubles at each level. Since this forms a geometric progression, the total parameter count of the `down` blocks is

$$18\,\text{base}^2\left(4^{\text{depth}}-1\right) \;+\; 8\,\text{base}\left(2^{\text{depth}}-1\right).$$

Similarly, in the upsampling path, the number of channels follows the reverse geometric sequence. Summing the contributions gives

$$\frac{35}{3}\,\text{base}^2\left(4^{\text{depth}}-1\right) \;+\; 5\,\text{base}\left(2^{\text{depth}}-1\right).$$

Finally, the output block contributes

$$9\,\text{out\_ch}\cdot\text{base} \;+\; 9\,\text{out\_ch}^2 \;+\; 4\,\text{out\_ch}.$$

Adding the contributions from all blocks yields the total number of parameters in the U-Net.

Substituting the values of `in_ch`, `out_ch`, `base`, and `depth` gives the final parameter count for the network.

From the above expression, upsampling block dominates the parameters count.

2. To compute the receptive field of the U-Net, we begin by analyzing the `DoubleConv` block. Each `DoubleConv` contains two 3×3 convolutions with stride 1. A single 3×3 convolution increases the receptive field by 2, so two such convolutions give a total increase of 4. Thus, the effective receptive field of the `DoubleConv` block is 5 (equivalent to a single 5×5 convolution). All other operations inside the block are pointwise and therefore do not affect the receptive field.

   We compute the receptive field by working backward from a single output pixel.

   The last convolution layer uses a 1×1 kernel, so the receptive field of the output pixel with respect to the previous feature map is 1.

   Next, we consider each of the `up` blocks. An `up` block consists of a transposed convolution with kernel size 2 and stride 2, followed by a `DoubleConv` block. Since the transposed convolution simply upsamples the feature map, it does not increase the receptive field. However, the subsequent `DoubleConv` increases the receptive field by 4.

   Since there are `depths` -1 such `up` blocks, the effective receptive field after all `up` blocks is

$$R_{\text{up}} \;=\; 1+4(\texttt{depths}-1).$$

   Contribution of the downsampling path: Given R_{up}, we now consider the effect of one down block. A down block contains:

   (a) a `DoubleConv` block, which increases the receptive field by 4, and

(b) a max-pooling layer with stride 2, which doubles the current receptive field.

Thus, the receptive field after one down block is

$$2\big(R_{\text{up}} + 4\big).$$

Repeating this over `depths` − 1 down-sampling levels gives

$$2^{\texttt{depths}-1} R_{\text{up}} + \sum_{i=1}^{\texttt{depths}-1} 2^i \cdot 4 = 2^{\texttt{depths}-1} R_{\text{up}} + 2^{\texttt{depths}+2} + 8.$$

Finally, after reaching the input resolution, the last `DoubleConv` block increases the receptive field by an additional 4.

Combining all contributions, the total receptive field of the U-Net is

$$R = 2^{\texttt{depths}-1}\big(4\,\texttt{depths} + 3\big) + 2^{\texttt{depths}+2} + 12.$$

3. Increase the Dilation factor for each layer
4. No
5. If replaces, each block as only linear operations, thus the U-Net turns into one large convolutional filter.
6. Its is not due to the max pooling layers, its translation invariant if we consider large translation which is a function of the depth.

Coding

Exercise 5 (Audio Denoising with Wiener and CNNs - ⋆⋆)**.** In this coding exercise we will use two strategies for audio denoising: a simple Wiener filter and a convolutional neural network. We'll hear that the CNN does better, but a Wiener filter will also be okay given that we will show it much less data: we will only use a single noisy audio clip and some magic.

In both cases we will use the discrete Fourier transform (DFT) to pass to the frequency domain, but instead of applying the DFT to the entire length of the signal we will split the audio into (overlapping) frames and apply the transform to the frames. This is called the (discrete) short-time Fourier transform (STFT). Using a single huge DFT of the entire length of the audio signal has a number of drawbacks. The main one is that the DFT tells us what frequencies are present in the signal, but not where they are. For audio that matters. This is why the DFT is computed on short overlapping frames: then we at least know that the frequencies belong to a given frame. Another benefit is that we can use the same processing for arbitrarily long audio signals since the frame size is fixed.

We will use the dataset available at `https://datashare.ed.ac.uk/handle/10283/2791`. This dataset contains clean speech audio files and versions corrupted with different types of noise at different signal-to-noise ratios (SNRs)—not only additive white Gaussian noise. Make sure to listen to a couple of samples to get a feel for the data.

STFT processing Let $x[n]$ be a single-channel (mono) audio clip of length N (and thus duration N/f_s with f_s the sampling frequency). As mentioned in the exercise introduction, we'll split x into overlapping frames of length M_w with hop size H (the number of samples between the starts of consecutive frames). This is illustrated in Figure 1.

The number of frames is

$$K = \left\lceil \frac{N - M_w}{H} \right\rceil + 1.$$

Each frame is multiplied with a "window" $w[m]$, $m = 0, \ldots, M_w - 1$ which rises and falls gradually; see Fig. .2. You do not have to worry about this since we use a ready-made implementation of STFT (the windowing prevents abruptly cutting off at the frame boundaries, which

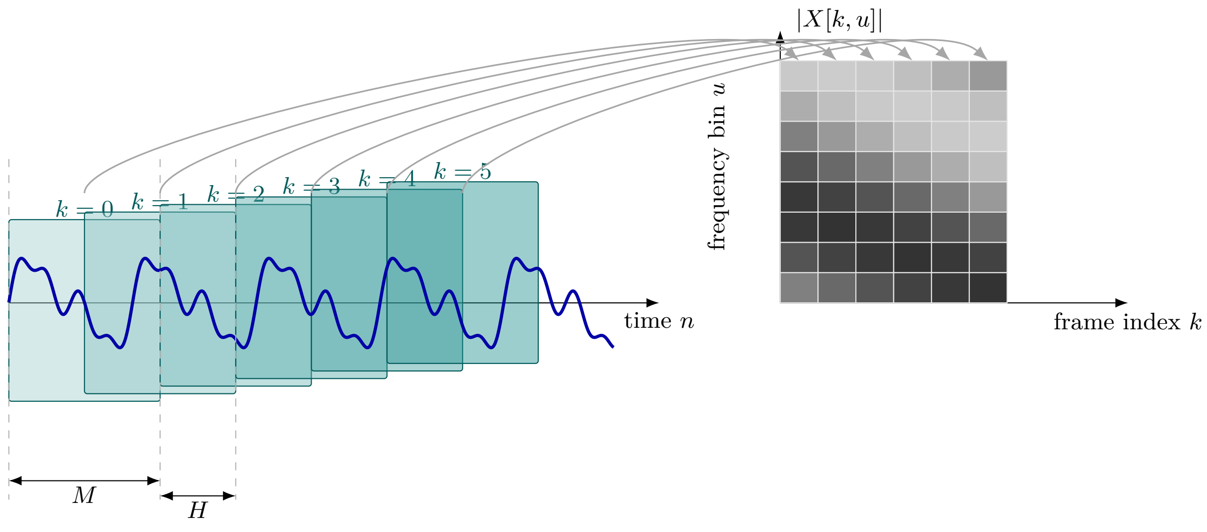

Figure 1: Short-time Fourier transform (STFT): overlapping windows of length M_w slide with hop size H across a signal. Each frame k yields a spectrum across frequency bins u, forming the time–frequency representation $X[k,u]$.

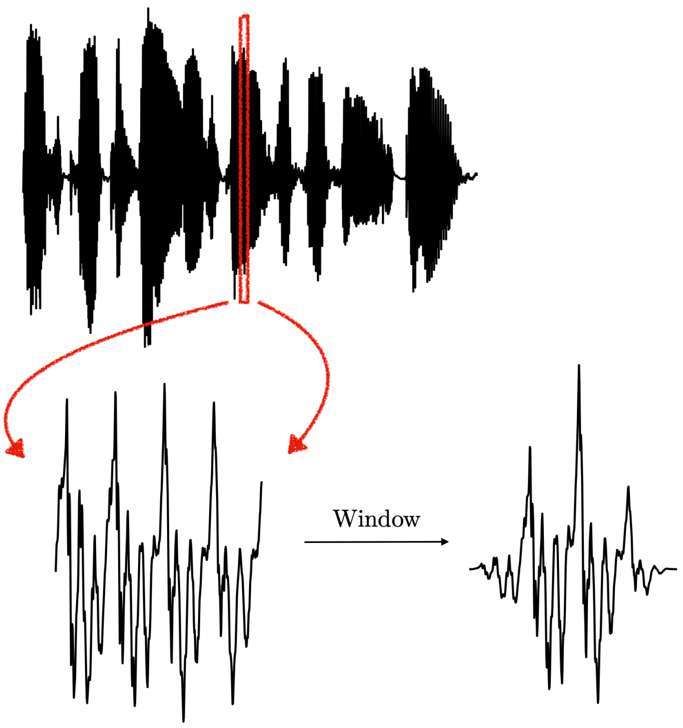

Figure 2: Effect of windowing a speech sample. We extract a 512 length sample segment from a clean speech signal and apply a Hann window, which tapers the edges to reduce spectral leakage.

among other things helps eliminate some audible artifacts). The short-time Fourier transform (STFT) of $x[n]$ is then defined as

$$X[k,u] = \sum_{m=0}^{M_w-1} x[m+kH]w[m]e^{-j2\pi um/N_{\text{fft}}}, \quad k=0,\dots,K-1, \quad u=0,\dots,N_{\text{fft}}-1$$

with $M_w \le N_{\text{fft}}$. Here, $X[k,u]$ is the STFT coefficient for frame k and frequency bin u. It is really just a DFT of the samples of x with indices from kH to $kH+M_w-1$, multiplied by the window $w[m]$.

A simple two-step inverse STFT (iSTFT) is:

$$\widetilde{x}_k[m] = \frac{1}{N_{\text{fft}}} \sum_{u=0}^{N_{\text{fft}}-1} X[k,u]\, e^{j2\pi um/N_{\text{fft}}}, \quad m=0,\dots,M_w-1,$$
$$\widehat{x}[n] = \frac{1}{C} \sum_{k=0}^{K-1} w[n-kH]\, \widetilde{x}_k[n-kH], \quad \text{with } 0 \le n-kH < M_w,$$

where the window w satisfies the so-called constant-overlap-add (COLA) condition (i.e., $\sum_k w^2[n-kH] = C$ is constant in n). The first line simply inverts the DFT on each frame; the second line overlaps and adds the frames back together to form the reconstructed signal $\widehat{x}[n]$. You don't need to know these details about the STFT on the exam. We are giving them here for completeness. You do need to understand what is going on here conceptually.

STFT turns 1D audio into a 2D time–frequency representation which can be processed as an image. Since STFT is complex valued, we often only work on the magnitude spectrogram $|X[k,u]|$. An example is in Figure 3. This is exactly what we will do with the CNN below—we will treat $|X[k,u]|$ as an image and process it by 2D convolutions.

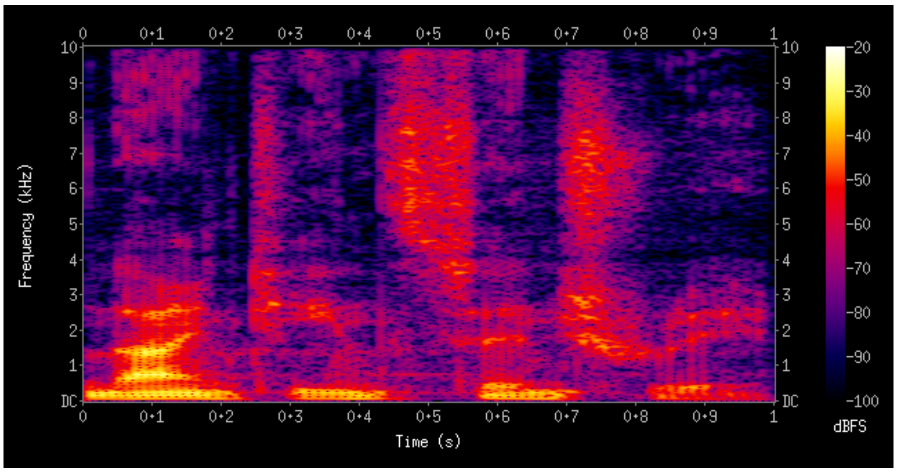

Figure 3: Spectrogram of the spoken words "nineteenth century". Frequencies are shown increasing up the vertical axis, and time on the horizontal axis. The legend to the right shows that the color intensity increases with the magnitude (power). Work by Aquegg from `https://en.wikipedia.org/wiki/Spectrogram#/media/File:Spectrogram-19thC.png`.

Wiener filter The first task is to denoise audio with a Wiener filter in the STFT domain using one gain per frequency, applied to every frame. We assume additive noise $y[n] = x[n] + w[n]$ with $X = \text{STFT}(x), Y = \text{STFT}(y), W = \text{STFT}(w)$. Just for fun we will assume that we only have access to the noisy signal y and its STFT Y. This means that we should estimate the per-frequency power spectral densities (PSDs) from the noisy clip alone by aggregating across frames k, then form the Wiener gain $G[u]$.

As in lecture, we'll use the formula

$$G[u] = \frac{\text{SNR}[u]}{1 + \text{SNR}[u]} = \frac{S_{\mathsf{xx}}[u]}{S_{\mathsf{xx}}[u] + S_{\mathsf{ww}}[u]}.$$

To estimate $S_{\mathsf{ww}}[u]$ we will use the fact that speech is relatively "sparse" in the STFT representation. This means that many $[k, u]$ bins $X[k, u]$ of speech are very small and there are only a handful of large coefficients. We have seen something similar when summing up the frequency components of an image starting with the largest ones (which were mostly low frequencies). This means that for many frequency bins u, the lower percentiles of the magnitudes $|Y[k, u]|$ across frames k will correspond to noise-only coefficients. We can thus estimate the noise PSD using a quantile estimator and then use the average power to estimate the noisy signal PSD. Your task is to implement this Wiener filter in the provided Python files.

Here is how it works: In the provided python file `wiener.py`, implement the following steps in the function `wiener_enhance_freqavg()`: For each frequency u:

- Collect the magnitudes $|Y[k, u]|$ across all frames k.
- Estimate the noise PSD as $\widehat{S}_{\mathsf{ww}}[u] = (\text{quantile}_k\{|Y[k, u]|\})^2$; (e.g., $q = 10\%$).
- Estimate the noisy signal PSD $\widehat{S}_{\mathsf{yy}}[u]$ as the average power across frames:

$$\widehat{S}_{\mathsf{yy}}[u] = \frac{1}{K} \sum_{k=0}^{K-1} |Y[k, u]|^2.$$

- Estimate the clean signal PSD as $\widehat{S}_{\mathsf{xx}}[u] = \max\{\widehat{S}_{\mathsf{yy}}[u] - \widehat{S}_{\mathsf{ww}}[u], 0\}$. This is motivated by the formula $S_{\mathsf{yy}} = S_{\mathsf{xx}} + S_{\mathsf{ww}}$ which holds for additive noise uncorrelated with the signal.

- Compute the Wiener filter $G[u] = \widehat{S}_{\mathsf{xx}}[u]/(\widehat{S}_{\mathsf{xx}}[u] + \widehat{S}_{\mathsf{ww}}[u])$ and apply this same $G[u]$ to all frames: $\widehat{S}[k,u] = G[u]Y[k,u]$.

Finally, invert with iSTFT. Use the provided STFT wrappers `stft(x, ...)` and `istft(X, ...)` from `audio_helper.py`. Please do not change its signature or return type. Return a 1-D torch.Tensor at 16 kHz, same dtype and device as the input. For numerical stability, use a small `eps` in divisions. An optional `gain_floor` may be applied by clamping G from below.

When you are done with implementing your `wiener_enhance_freqavg()` function, you can apply your Wiener filter to a few noisy clips and listen to the results. You can listen and sanity-check your result by running `eval_wiener_and_export(...)` in `audio_denoising.py`, which calls your `wiener_enhance_freqavg` and prints the SNR improvement while saving a few WAVs to a folder ”out”. You will hear that sometimes it works quite well and sometimes it leaves a lot to be desired. For what kind of noise does it work well? How is this related to the assumptions used to estimate the noise PSD? Try playing with the quantile q used for noise PSD estimation.

Convolutional neural network We will now compare this Wiener filter with a very simple convolutional neural network (CNN). The CNN is provided as `TinyDenoiser` in `models.py`, with training and evaluation entry points `train_cnn_denoiser(...)` and `eval_cnn_and_export(...)` in `audio_denoising.py`. You do not need to modify the CNN; it is there to let you compare to the Wiener result and to listen to outputs.

This is what the code does: The CNN will take as input the magnitude spectrogram $|Y[k,u]|$ of the noisy audio and output an estimate of the clean magnitude spectrogram $|\widehat{X}[k,u]|$. The network treats the magnitude spectrogram as a single-channel image, where the time frames correspond to one spatial dimension and the frequency bins to the other spatial dimension. The phase of the noisy audio $Y[k,u]$ will be reused for reconstruction; the CNN does not use it at all. The CNN architecture is very simple: it consists of a few convolutional layers with ReLU activations, followed by a final convolutional layer with a sigmoid activation to ensure that the output magnitudes are non-negative.

Again, use the provided code to apply the learned non-linear filter to a couple of noisy clips and listen to the results. How do they compare with the Wiener filter? This is a simple version of the kind of audio processing implemented in the devices you're using every day!