Study Notes for Image Analysis, 2017

This document contains study notes for the course in Image Analysis given at Lund University, 2017. The notes are still work in progress, so read with caution. Errors and suggestions of improving the notes are much appreciated. Just send me an e-mail and I’ll try to fix it.

Students following the course have different background. For some the mathematics is going to be a challenge, for other the programming is more difficult. The course contain core competences that everyone is supposed to acquire. These are covered in the take-home assignments. Successful completion of all take-home assignments is a requirement for a pass (grade 3) on the course.

All material is posted on the course homepage:
http://www ctr.maths.lu.se/course/newimagean/2017/

Image analysis has both theoretical and practical components. One skill that you will work on is to understand the connection between theory (an equation, a theorem, a description of a method) and practical code. Understanding how to take a theoretical concept and turning it to a working program is not at all easy, but when doing so one gains a deeper understanding of the theory.

Credits: The image analysis course was developed by Gunnar Sparr in 1988. Since then the course has been taught in whole or in part by several researchers, e.g. Anders Heyden, Fredrik Kahl, Magnus Oskarsson, Jan-Erik Solem, Kalle Åström, Håkan Ardö, Mikael Nilsson, Carl-Gustav Werner and many more. The course material is a product of all of these authors.

In particular the review of linear algebra, the fourier transform and convolution (Lectures 3-4) were written in Swedish by Gunnar Sparr. Here I have tried to translate it to English, with some minor additions.

For the course we have used several textbooks, e.g. Digital Image Processing by Rafael C. Gonzales and Richard E. Woods.


Computer Vision: A Modern Approach by David A. Forsyth and Jean Ponce.

Also for the presentation material, slides etc. we have re-used slides from numerous researchers and from other computer vision courses. We have tried to write out the correct slide credits on the appropriate slides.
Chapter 1

Lecture 1: Introduction, Image models, Discrete Geometry, Gray-Level Transformations

1.1 Introduction

Read chapter 1 in Szelisky. This contains an overview on what image analysis and computer vision is, why it is useful and contains numerous examples of computer vision and image analysis applications.

1.2 Image Models

It is common to use both continuous and discrete image models. Both models are useful for intuition and both have merits in different applications, for algorithms and for better understanding of image analysis. Read chapter 2.3 in Szelisky. It contains a bit on sampling. It also contains more information about colour images, that I usually don’t have time to talk about during the lectures.

For the **continuous model** images are thought of as functions from a region $\Omega \subset \mathbb{R}^2$, typically we model images as functions to $\mathbb{R}$, i.e. $f : \Omega \rightarrow \mathbb{R}$, but sometimes we also work with complex images $f : \Omega \rightarrow \mathbb{C}$. For colour images there are typically three colour bands, such images would be modelled as $f : \Omega \rightarrow \mathbb{R}^3$. Multispectral images could have even more spectral channels, $f : \Omega \rightarrow \mathbb{R}^k$, where $k$ denotes the number of spectral channels. When working with fourier transforms we again use complex images, so a multi-channel complex image could be thought of as a function $f : \Omega \rightarrow \mathbb{C}^k$.

For the **discrete model** images are thought of as functions from a region $\Omega \subset \mathbb{Z}^2$. This means that the input space (or domain) is discretized. This process is called **sampling**. In practice the output space (or range) is also discretized. This is usually called **quantization**. Coarser sampling leads to checkerboard like images. Coarser quantization usually leads to false contours.

The range is often chosen as a power of 2, e.g. many gray-scale images have $256 = 2^8$ gray-levels, so called 8 bit grayscale images. This means that one needs 8 bits to store one such value. Thus discrete images are often thought of as functions $f : \Omega \rightarrow \{0, \ldots, 2^8\} \subset \mathbb{Z}$, where $\Omega \subset \mathbb{Z}^2$.

However, once such images are loaded into a computer, and we start working with it we often consider discrete the domain as either $\mathbb{R}$ or $\mathbb{C}$.

OK. this might be confusing. The main point is that images can be thought of as functions. To simplify one may think of **continuous model images** as

$$f : \mathbb{R}^2 \rightarrow \mathbb{R}$$

and **discrete model images** as

$$f : \mathbb{Z}^2 \rightarrow \mathbb{R}.$$
Take a look at 
http://www.ctr.maths.lu.se ... gettingstartedwithimages.html
which contains a few introductory matlab-commands for reading and viewing images.

Remark. Without going into much detail on function spaces and such. For those of you that have studied linear systems, yet another way to think of a discrete image \( f \) is to model the image as a distribution \( F_\delta \),

\[
F = \sum_i \sum_j f(i,j) \delta(x - i, y - j).
\]

For finite images, there are only a finite number of pixels, so we usually consider the function to be zero outside the width of the image.

1.3 Interpolation and Discretization

The idea of discretization or sampling is to find a way to map a function \( F : \mathbb{R}^2 \to \mathbb{R} \) to one defined only on the integer points, i.e. \( f : \mathbb{Z}^2 \to \mathbb{R} \). The idea of interpolation is to get back, i.e. to take a function \( f : \mathbb{Z}^2 \to \mathbb{R} \) as input and produce a function \( F : \mathbb{R}^2 \to \mathbb{R} \).

Let \( \mathcal{B} \) denote the set of continuous functions \( F : \mathbb{R}^2 \to \mathbb{R} \) that are also square integrable, i.e. such that the integral

\[
\int_{x=-\infty}^{\infty} \int_{y=-\infty}^{\infty} |F(x,y)|^2 dx dy
\]

exists and such that the Fourier transform is zero outside the interval \([-\pi, \pi] \times [-\pi, \pi]\).

Let \( \mathbb{L}^2 \) denote the set of discrete functions \( f : \mathbb{Z}^2 \to \mathbb{R} \) that are also square integrable, i.e. such that the sum

\[
\sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} |f(i,j)|^2
\]

exists.

For such functions introduce the discretization operator \( D : \mathcal{B} \to \mathbb{L}^2 \), such that

\[
f(i,j) = D(F)(i,j) = F(i,j).
\]

The normalized sinc function, \( \text{sinc} \), is defined as

\[
\text{sinc}(x) = \begin{cases} 
\frac{\sin \pi x}{\pi x}, & x \neq 0 \\
1, & x = 0
\end{cases}
\]

Introduce the ideal interpolation operator \( I : \mathbb{L}^2 \to \mathcal{B} \), such that

\[
F(x,y) = I(f)(x,y) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} \text{sinc}(x-i) \text{sinc}(y-j) f(i,j).
\]

Then we have that ideal interpolation restores the sampled function, i.e.

\[
I(D(F)) = F.
\]

Other interpolation methods can often be written in a similar way. Introduce

\[
F_h(x,y) = I_h(f)(x,y) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} h(x-i,y-j) f(i,j).
\]

With

\[
h(x,y) = \begin{cases} 
1, & -\frac{1}{2} \leq x \leq \frac{1}{2}, -\frac{1}{2} \leq y \leq \frac{1}{2} \\
0, & \text{otherwise}
\end{cases}
\]
we get piece-wise constant interpolation. With

\[ h(x, y) = \frac{1}{2\pi \sigma^2} e^{-\left(x^2+y^2\right)/(2\sigma^2)} \]

we get gaussian interpolation, which can be understood as an approximation to a combination of ideal interpolation and Gaussian smoothing. But more on this after the lectures on convolution and scale-space.

Read chapter 2.3.1 in Szelisky. Also look at

http://www.ctr.maths.lu.se ... interpolation1d.html

which contains a matlab-illustration to interpolation of functions of one variable and

http://www.ctr.maths.lu.se ... interpolation.html

which contains a matlab-illustration to interpolation of functions of two variables.

1.4 Discrete Geometry:

Read chapter 3.3 in Szelisky. It contains distance transforms and connected components, but also morphology, which we will talk about later in the course.

1.5 Gray-level transformations

Read chapter 3.1 in Szelisky. It contains both gray-level transformations and histogram equalization. Also take a look at the example:

http://www.ctr.maths.lu.se ... histeqdemo.html
Chapter 2

Lecture 2: Machine Learning and Pixelwise Classification

2.1 Classification examples

One indispensable tool for image analysis is that of machine learning. It is a tool that is used in different components, but also as a whole. This lecture contains the first step, but as we shall see it can be used already for generating gray-level transforms.

Example 2.1.1. Assume that a pixel values \( f(i, j) \) is given, classify the pixel as heart or not-heart.

Example 2.1.2. OCR: Given an image, segment the characters and classify them (a, b, c, etc).

Cell analysis: Given an image, segment the cells and classify them (white blood cell, red blood cells, etc)
Grain analysis: Given an image of grain, segment the grains and classify them (wheat, rice, stone, rotten, etc)
Handwriting recognition: Given handwritten data, segment the text in individual characters and classify them (a, b, c, etc),
Diagnostic support: Given medical data (images, journal data) classify patient (healthy, sickness1, sickness2, etc)
Face detection: Given an image, classify it as (face, not face)

All of these classification problems have in common:

- data - \( x \)
- A number of classes

One would like to determine a class for every possible feature vector.

Here we will assume that the features are represented as a column vector, i.e. \( x \in \mathbb{R}^n \),

\[
x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}
\]

In fact to begin with (for simplicity) we will assume that there is only one feature, i.e. that \( x \in \mathbb{R} \).

One would like to compare the feature vector \( x \) with those that one usually gets with a number of classes. Let \( y \) denote the class index, i.e. the classes are \( y \in \omega_y = \{1, \ldots, M\} \) where \( M \) denotes the number of classes.

Let’s start with a simple toy example to understand the basic idea and to define some notation.

Example 2.1.3. In a small town, there are two bicycle brands albatross and butterfly. Albatross sell mostly green bikes (80 percent) are green and the rest are yellow. Butterfly sell 50 percent green bikes and 50 percent yellow. The Albatross brand is more popular. They have 80 percent of the market in the town. In another nearby town, from a distance you see a yellow bike. What is the probability that the bike is an Butterfly bike?
How is the example related to problems in image analysis?

**Example 2.1.4.** In a medical image, there are two types of pixels background and heart. Background pixels are usually dark (80 percent) but sometimes bright. Heart pixels are usually bright (50 percent) of the time. Background pixels are more common. In a usual medical image 80 percent of the pixels are background. You see a bright pixel. What is the probability that this is a heart pixel?

Assume that one feature vector (measurements) $x$ and class (the unknown quantity) $y$ are drawn from a joint probability distribution. If one can calculate the probability that the class is $y$ given the measurements $x$, i.e. the so called **posterior probability**

$$P(y|x)$$

then it makes sense to classify the measurement according to the class that gives the highest posterior probability.

The joint probability $P(x, y)$ is related to the prior probability as

$$P(x) = \sum_y P(x, y)$$

and to the total probability

$$P(y) = \sum_x P(x, y).$$

The joint probability $P(x, y)$ is related to the prior probability $P(y)$ and the conditional probability $P(x|y)$ according to $P(x, y) = P(x|y)P(y)$, but also as $P(x, y) = P(y|x)P(x)$

**Example 2.1.5.** In a small town, there are two bicycle brands albatross and butterfly. Albatross sell mostly green bikes (80 percent) are green and the rest are yellow. Butterfly sell 50 percent green bikes and 50 percent yellow. The Albatross brand is more popular. They have 90 percent of the market in the town. In another nearby town, from a distance you see a yellow bike. What is the probability that the bike is an Butterfly bike? The prior probability of the two brands are

$$P('Albatross') = 0.9,$$

$$P('Butterfly') = 0.1.$$

So if you see a bike, but cannot see the colour or the brand, there is a 90 percent chance that it is an 'Albatross' bike. This is the prior probability of the two unknown classes before getting any new measurements $x$.

The measurement probabilities are

$$P('green'|'Albatross') = 0.8,$$

$$P('yellow'|'Albatross') = 0.2,$$

$$P('green'|'Butterfly') = 0.5,$$

$$P('yellow'|'Butterfly') = 0.5$$

The Joint probabilities are

$$P('green', 'Albatross') = 0.8 \cdot 0.9 = 0.72$$

$$P('yellow', 'Albatross') = 0.2 \cdot 0.9 = 0.18$$

$$P('green', 'Butterfly') = 0.5 \cdot 0.1 = 0.05$$

$$P('yellow', 'Butterfly') = 0.5 \cdot 0.1 = 0.05$$

The total probabilities of the two measurements are

$$P('green') = 0.8 \cdot 0.9 = 0.72 + 0.05 = 0.77$$

$$P('yellow') = 0.2 \cdot 0.9 = 0.18 + 0.05 = 0.23$$
The a posteriori probabilities are

\[
P(\text{'Albatross'|'green'}) = \frac{0.72}{0.77} = 0.928571429
\]
\[
P(\text{'Albatross'|'yellow'}) = \frac{0.18}{0.23} = 0.787234043
\]
\[
P(\text{'Butterfly'|'green'}) = \frac{0.05}{0.77} = 0.064951613
\]
\[
P(\text{'Butterfly'|'yellow'}) = \frac{0.05}{0.23} = 0.217391304
\]

Before seeing the colour of the bike, the a priori probability of it being a 'Butterfly' bike is only 5 percent. After seeing that it is yellow the probability has increased to 22 percent. But if you had to guess which brand it is, the best guess is still that it is an 'Albatross' bike.

Understanding the classification problem often boils down to

1. modelling the a priori probabilities \( P(y) \), modelling and then estimating the measurement probabilities \( P(x|y) \).

2. It is then straightforward to calculate the joint probabilities

\[
P(x, y) = P(x|y)P(y)
\]

3. From the joint probabilities one can calculate the total probabilities

\[
P(x) = \sum_y P(x, y)
\]

4. and then the a posteriori probabilities

\[
P(y|x) = \frac{P(x, y)}{P(x)}
\]

5. The best guess of the unknown class \( y \) is the one that has the highest a posterior probability \( P(y|x) \)

**Remark.** For a given measurement \( x \), the total probability \( P(x) \) is constant. Finding the best guess \( y \) can be found by maximizing the joint probability \( P(x, y) \).

**Remark.** Both measurements \( x \) and classes \( y \) might be discrete or continuous. If they are continuous, similar formulas are obtained by replacing probabilities with probability density functions.
Chapter 3

Lecture 3: Linear Algebra and the Fourier Transform

3.1 Overview

In this lecture we show that images can be thought of as objects of a linear vector space. There are many useful tools from linear algebra that can be used to analyse and process images.
In the lecture we also talk about the two-dimensional fourier transform, which we use for analyzing images.
Read more about this in chapter 3.4 in Szelisky.
For a review on linear algebra you could also try our new interactive book
http://www.immersivemath.com

3.2 Linear Algebra

The first half of the lecture is a review on linear algebra and examples on how this can be used in image analysis.

Vector spaces $\mathbb{R}^n$ and $\mathbb{C}^n$

A vector space is a set with certain operations and properties. With $\mathbb{R}^n$ is meant the set of $n \times 1$–matrices,

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

where $x_i \in \mathbb{R}$

Similarly with $\mathbb{C}^n$ is meant matrices where the elements are complex, i.e. $x_i \in \mathbb{C}$.

For vectors we define addition as elementwise addition and define multiplication with a scalar as multiplying the scalar with every element in the vector. These operations obey certain laws as described in
See Definition 2.11 on
http://immersivemath.com/ila/ch02vectors/ch02.html

$\mathbb{R}^n$ and $\mathbb{C}^n$ are examples of linear spaces, or vector spaces. For the vector space $\mathbb{R}^3$ the elements can be thought of as geometric vectors.
A base in $\mathbb{C}^n$ (or $\mathbb{R}^n$) is a set of vectors $\{e_1, \ldots, e_n\}$ such that

(i) the elements are linearly independent

(ii) each vector $x \in \mathbb{C}^n$ (or $\mathbb{R}^n$) can be written

$$x = \xi_1 e_1 + \cdots + \xi_n e_n$$

$\xi_i \in \mathbb{C}$ (resp $\mathbb{R}$)
The natural or canonical basis in $\mathbb{C}^n$ consists of the following basis elements

$$
e_1 = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad e_2 = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \quad \ldots, \quad e_n = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}.$$

Let $A$ be a matrix, whose elements are complex. For such matrices we use the following notation $A^* = A^T$ to represent both transpose and taking complex conjugate. For real matrices $A^* = A^T$.

**Definition 3.2.1.** The scalar product of $x$ and $y$ is defined as

$$x \cdot y = x^*y = \sum_{i=1}^n \bar{x}_i y_i.$$

Note that $y \cdot x = \bar{x} \cdot \bar{y}$. Again note that for real vectors, $x, y \in \mathbb{R}^n$ there is no need for taking the complex conjugate. Therefore $x \cdot y = y \cdot x$.

**Definition 3.2.2.** For vectors in $\mathbb{C}^n$ and $\mathbb{R}^n$ we define

- **Orthogonality** $x \perp y \iff x \cdot y = 0$
- **Length, norm** $||x|| = \left(\sum_{i=1}^n \bar{x}_i^2\right)^{1/2} = (x \cdot x)^{1/2}$

**Theorem 3.2.1.** (Pythagoras.)

$$x \perp y \iff ||x + y||^2 = ||x||^2 + ||y||^2$$

**Proof.**

$$||x + y||^2 = (x + y)^*(x + y) = x^*x + x^*y + y^*x + y^*y = x^*x + y^*y = ||x||^2 + ||y||^2$$

**Definition 3.2.3.** $\{e_1, \ldots, e_n\}$ is said to be an orthonormal basis in $\mathbb{C}^n$ (resp $\mathbb{R}^n$) if

$$e_i \cdot e_j = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases}$$

**Theorem 3.2.2.** Assume that $\{e_1, \ldots, e_m\}$ is an orthonormal basis, $x = \sum_{i=1}^n \xi_i e_i$. Then

$$\xi_i = e_i \cdot x = e_i^*x$$

$$||x||^2 = \sum_{i=1}^n |\xi_i|^2$$

**Proof.**

$$x = \sum_{i=1}^n \xi_i e_i \implies e_i \cdot x = e_i \cdot \sum_{i=1}^n \xi_i e_j = \sum_{i=1}^n \xi_j e_i \cdot e_j = \xi_i$$

\[\square\]
3.2.1 Projection onto a subspace

For three dimensional vectors, projection onto a plane can be illustrated in the following figure.

\[ \text{The shortest distance from } u \text{ to the plane } \pi \text{ is given by the orthogonal distance} \]
\[ \min_{w \in \pi} ||u - w|| = ||u - u_\pi|| \text{ därfür } u_\pi = u : \text{s projection on } \pi. \]

The projection \( u_\pi \) is characterized by
\[ u_\pi \in \pi \]
\[ u - u_\pi \perp w \text{ for every } w \in \pi \]

This is true in the general case. Given vectors \( \{a_1, \ldots, a_k\} \subset \mathbb{C}^n \text{ or } \mathbb{R}^n, \ k \leq n. \) Let
\[ A = [a_1 \ldots a_k] \text{ be a } n \times k \text{ matrix with columns } a_1, \ldots, a_k, \]
\[ \pi = \{w \mid w = \sum_{i=1}^{n} x_i a_i = Ax \text{ where } x_i \in \mathbb{C} \text{ (or } \mathbb{R})\} \]

**Lemma 3.2.1.** Pseudo If \( \{a_1, \ldots, a_k\} \) are linearly independent in \( \mathbb{C}^n \text{ (or } \mathbb{R}^n) \) then \( A^* A \) is invertible.

**Proof.**
\[ 0 = A^* Ax \implies 0 = x^* A^* A x = (Ax)^* (Ax) = ||Ax||^2 \implies \]
\[ 0 = Ax = \sum_{i=1}^{k} x_i a_i \implies x = 0 \]

where the last implication follows since \( \{a_1, \ldots, a_k\} \) are linearly independent. This proves that \( A^* A \) is invertible.

We define projection onto a subspace in \( \mathbb{C}^n \text{ or } \mathbb{R}^n \) as follows.

**Definition 3.2.4.** \( u_\pi \) is said to be the projection of \( u \) onto \( \pi \) if
\[ \begin{cases} u_\pi \in \pi \\ u - u_\pi \perp w \text{ for every } w \in \pi \end{cases} \]

The projection is characterized by the following theorem:
Theorem 3.2.3. Assume that the columns of $A$ are linearly independent. Then

(i) The projection of $u$ onto $\pi$ is given by

$$u_{\pi} = x_1a_1 + \ldots + x_ka_k = Ax \quad \text{with} \quad x = (A^*A)^{-1}A^*u$$

(ii) $\min_{w \in \pi}||u - w|| = ||u - u_{\pi}||$

Proof. (i) $u_{\pi} = Ax$ projection of $u$ onto $\pi \iff$

\[
\begin{align*}
  & a_1^*(u - u_{\pi}) = 0 \\
  & \quad \vdots \\
  & a_k^*(u - u_{\pi}) = 0
\end{align*}
\]

$\iff A^*(u - Ax) = 0 \iff A^*u = A^*Ax \iff x = (A^*A)^{-1}A^*u$

(ii) Pythagoras theorem gives

$$u - w = u - u_{\pi} + u_{\pi} - w \implies ||u - w||^2 = ||u - u_{\pi}||^2 + ||u_{\pi} - w||^2 \geq ||u - u_{\pi}||^2$$

with equality if $\iff w = u_{\pi}$. $\square$

Definition 3.2.5. $A^+ = (A^*A)^{-1}A^*$ is called the pseudo inverse of $A$.

According to theorem 3.2.3, the projection $u_{\pi}$ of $u$ onto $\pi$ is obtained by $u_{\pi} = A^+u$. The pseudo inverse can also be defined if the columns of $A$ are not linearly independent. For the case of orthogonal columns, the projection is particularly simple.

Theorem 3.2.4. If \{a_1, \ldots, a_k\} are orthogonal then

$$u_{\pi} = y_1a_1 + \ldots + y_ka_k \quad \text{med} \quad y_i = a_i \cdot u = a_i^*u, \quad i = 1, \ldots, k.$$  

Proof. If \{a_1, \ldots, a_k\} are orthonormal then $A^*A = I$ (Unit matrix of size $k \times k$). Therefore $u_{\pi} = A(A^*A)^{-1}A^*u = AA^*u = Ay$ with $y_i = a_i^*u$. $\square$

### 3.2.2 Projection onto an affine subspace

Given vectors \{a_1, \ldots, a_k\} $\in \mathbb{C}^n$ or $\mathbb{R}^n$, $k \leq n$, the set

$$\pi = \{w \mid w = \sum_{i=1}^{n} x_i a_i = Ax \quad \text{where} \quad x_i \in \mathbb{C} (\text{or} \mathbb{R})\}$$

is a subspace and contains the zero vector by definition. In the previous section we discussed projection onto such a subspace.

Sometimes it is of interest to study projection onto an affine subspace of type

$$\pi = \{w \mid w = m + \sum_{i=1}^{n} x_i a_i = Ax + m \quad \text{where} \quad x_i \in \mathbb{C} (\text{or} \mathbb{R})\}.$$  

Notice that this is not necessarily a subspace since it might not contain the zero vector. Affine subspaces are characterized by displacement vectors being a linear space.

An affine subspace can be defined by one point $m$ of the space and a set of $k$ vectors $a_1, \ldots, a_k$. The projection of $u$ onto $u_{\pi}$ on the affine subspace is similar to projection onto an ordinary subspace. The difference is only that $m$ is subtracted from $u$ first and then added after the projection.

- Subtract $m$, i.e. form $v = u - m$.
- Project $v$ onto the space spanned by $a_1, \ldots, a_k$, i.e. $v_{\pi} = A^+v$.
- Add $m$, i.e. form $u_{\pi} = v_{\pi} + m$. 

11
3.2.3 Principal Component Analysis

In the previous two sections we talked about projection onto a subspace (defined by vectors $a_1, \ldots, a_k$) or onto an affine subspace (defined by vectors $m, a_1, \ldots, a_k$. But how can we determine a suitable subspace from examples.

The key idea here is that given many examples $x_1, \ldots, x_N \in \mathbb{C}^n$ or $\mathbb{R}^n$ find an subspace or affine subspace $\pi$ so that the errors when projecting all of the examples are small in some sense. The calculations become particularly easy if we choose a particular error.

Assume that an affine subspace $\pi = \{ w \mid w = m + \sum_{1}^{n} x_i a_i = Ax + m \text{ where } x_i \in \mathbb{C} \text{ (or } \mathbb{R}) \}.$ of dimension $k$ is defined by vectors $m, a_1, \ldots, a_k$. Assume that the orthogonal projection of example $x_i$ onto the affine subspace $\pi$ is denoted $y_i(\pi)$. Introduce the following error

$$e(\pi) = \sum_{i=1}^{N} ||y_i(\pi) - x_i||^2.$$

The affine subspace $\pi$ that minimizes $e(\pi)$ can be found by the following method.

1. Calculate the mean $m = \frac{1}{N} \sum_{i=1}^{N} x_i$.
2. Subtract the mean from all examples $z_i = x_i - m$.
3. Place all of the resulting vectors as columns of a matrix, $M = (z_1 \ldots z_N)$.
4. Factorize $M$ using the singular value decomposition $M = USV^T$.
5. Use the first $k$ columns of $U$ as the basis of the subspace, i.e. $a_i = u_i$, with $U = (u_1 \ldots u_m)$.

3.2.4 Images as elements of a vector space

In linear algebra courses, the idea of vectors are often first introduced using geometric vectors. We define how to add two vectors $\vec{u}$ and $\vec{v}$ as well as how to multiply a scalar (number) with a vector $\vec{u}$. Then it is shown that these operations fulfill a number of properties

**Definition 3.2.6. Properties of Vector Arithmetic** Assuming that $\vec{u}$, $\vec{v}$, and $\vec{w}$ are vectors of the same type, and that $k$ and $l$ are scalars, then the following rules hold:

\[
\begin{align*}
(i) \quad & \vec{u} + \vec{v} = \vec{v} + \vec{u} \quad \text{(commutativity)} \\
(ii) \quad & (\vec{u} + \vec{v}) + \vec{w} = \vec{u} + (\vec{v} + \vec{w}) \quad \text{(associativity)} \\
(iii) \quad & \vec{v} + \vec{0} = \vec{v} \quad \text{(zero existence)} \\
(iv) \quad & \vec{v} + (-\vec{v}) = \vec{0} \quad \text{(negative vector existence)} \\
(v) \quad & k(\vec{v}) = (kl)\vec{v} \quad \text{(associativity)} \\
(vi) \quad & l\vec{v} = \vec{v} \quad \text{(multiplicative one)} \\
(vii) \quad & 0\vec{v} = \vec{0} \quad \text{(multiplicative zero)} \\
(viii) \quad & k\vec{0} = \vec{0} \quad \text{(multiplicative zero vector)} \\
(ix) \quad & k(\vec{u} + \vec{v}) = k\vec{u} + k\vec{v} \quad \text{(distributivity 1)} \\
(x) \quad & (k + l)\vec{v} = k\vec{v} + l\vec{v} \quad \text{(distributivity 2)}
\end{align*}
\]

(3.1)

AfterIntroducing a set of basis vectors $\vec{e}_1, \ldots, \vec{e}_n$ every vector can be identified by its coordinates $u_1, \ldots, u_n$, so that

$$\vec{u} = \sum_{i=1}^{n} u_i \vec{e}_i.$$
We often stack these coordinates in a column vector

$$u = \begin{pmatrix} u_1 \\ \vdots \\ u_n \end{pmatrix}$$

and obtain the usual rules for calculating addition and multiplication with a scalar.

This idea is much more general and useful.

**Definition 3.2.7. Vector space** A vector space consists of a set $V$ of objects (called vectors) and a field $F$, together with a definition of vector addition, $\mathbf{u} + \mathbf{v}$, and multiplication of a scalar with a vector, $k\mathbf{u}$, in such a way that the properties of 3.2.6 holds.

There are many interesting vector spaces, e.g. polynomials up to a fixed degree and functions $\mathbb{R} \rightarrow \mathbb{R}$.

**Example 3.2.1.** Polynomials in one variable of degree 2 is a vector space. One possible basis is

$$\mathbf{e}_1(x) = 1, \quad \mathbf{e}_2(x) = x, \quad \mathbf{e}_3(x) = x^2.$$ 

The polynomial $\mathbf{u}(x) = 5x^2 + 3x - 2$ has coordinates $u = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \begin{pmatrix} -2 \\ 3 \\ 5 \end{pmatrix}$, since

$$\mathbf{u} = -2 \mathbf{e}_1 + 3 \mathbf{e}_2 + 5 \mathbf{e}_3 = 5x^2 + 3x - 2.$$ 

The dimension of the vector space is 3.

**Example 3.2.2.** Matrices of size $2 \times 2$ is a vector space. One possible basis is

$$\mathbf{e}_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad \mathbf{e}_2 = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \quad \mathbf{e}_3 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \quad \mathbf{e}_4 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}.$$ 

The matrix

$$\mathbf{u} = \begin{pmatrix} 1 \\ 3 \\ 7 \\ 2 \end{pmatrix}$$ 

has coordinates $u = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 3 \\ 7 \\ 2 \end{pmatrix}$, since

$$\mathbf{u} = u_1 \mathbf{e}_1 + u_2 \mathbf{e}_2 + u_3 \mathbf{e}_3 + u_4 \mathbf{e}_4 = \begin{pmatrix} 1 \\ 3 \\ 7 \\ 2 \end{pmatrix}.$$ 

The dimension of the vector space is 4.

Think of this as a world where matrices with real elements are the objects $\mathbf{u}$) and a world where column vectors $u$ with real numbers are the objects and that there are methods for going back and forth between these two worlds. The conversion is made through a set of basis vectors, i.e.

$$\mathbf{u} = \sum_{i=1}^{n} u_i \mathbf{e}_i.$$
For images, we will in the sequel often use a particular method (column stacking) for converting between images and column vectors. A digital grayscale image can be represented by a matrix

\[
f = \begin{bmatrix}
  f(1,1) & f(1,2) & \ldots & f(1,N) \\
  f(2,1) & f(2,2) & \ldots & f(2,N) \\
  \vdots & \vdots & \ddots & \vdots \\
  f(M,1) & f(M,2) & \ldots & f(M,N)
\end{bmatrix}.
\]

Here we let the matrix elements \( f(i,j) \) be complex (or real) elements. We will use the following notation to denote row \( j \) and column \( k \).

\[
f(j,:) = [f(j,1) \ f(j,2) \ \ldots \ f(j,N)]
\]

\[
f(\cdot,k) = \begin{bmatrix}
  f(1,k) \\
  f(2,k) \\
  \vdots \\
  f(M,k)
\end{bmatrix}.
\]

Introduce the following notation for columnstacking of a matrix.

\[
\tilde{f} = \begin{bmatrix}
  f(\cdot,1) \\
  f(\cdot,2) \\
  \vdots \\
  f(\cdot,N)
\end{bmatrix}
\]

(In matlab the column stacked vector \( v \) can be obtained from a matrix \( f \) using \( v=f(:) \).

Since \( \tilde{f} + \tilde{g} = \tilde{f} + \tilde{g}, \tilde{f} = \lambda \tilde{f} \) we can identify complex (or real) \( M \times N \)-matrices with vectors of type \( \mathbb{C}^{MN} \) (or \( \mathbb{R}^{MN} \)).

This is a key idea: Images can be thought of a elements of a vector space. We can add two images, multiply them with a scalar, calculate the scalar product between two images and project an image onto a subspace spanned by a set of images.

It might be confusing to say that images are vectors. The word vector has several different meanings (i) a vector can mean a matrix of size \( M \times 1 \) or \( 1 \times N \), (ii) a vector can denote a geometric vector in three dimensions (iii) a vector can be an element of a linear vector space.

So start getting used to calling matrices ‘vectors’ (when you known what you’re doing). In a way we are finished here. Everything that we can do with vectors (addition, scalar multiplication, scalar product, coordinates, basis, orthogonal projection, projection onto affine subspace, principal component analysis) we can now do with images after first column-stacking them.

But for better understanding it might be useful to go through some of the material once more, specifically for image matrices. The natural (canonical) basis for \( M \times N \)-matrices consists of matrices

\[
\chi_{ij} = \begin{bmatrix}
  0 & \cdots & 0 & \cdots & 0 \\
  \vdots & \ddots & \vdots & \ddots & \vdots \\
  0 & \cdots & 1 & \cdots & 0 \\
  \vdots & \ddots & \vdots & \cdots & \vdots \\
  0 & \cdots & 0 & \cdots & 0
\end{bmatrix},
\]

which has zeros everywhere except at position \( i, j \), where there is a single one. The coordinates of the matrix \( f \) in this basis is the matrix elements (the pixel values)

\[
f = \sum_{i,j} f(i,j) \chi_{ij}.
\]
It is sometimes useful to change coordinates. This can be useful for example in image compression or for better understanding of a problem.
As usual the computations are substantially simpler when using an orthonormal basis.
The scalar product between two matrices is
\[ f \cdot g = \sum_{i=1}^{M} \sum_{j=1}^{N} \bar{f}(i,j)g(i,j). \]
The norm is defined using the scalar product, i.e.
\[ ||f|| = \sqrt{f \cdot f} = \sqrt{\sum_{i=1}^{M} \sum_{j=1}^{N} \bar{f}(i,j)f(i,j)}. \]

**Example 3.2.3 (Scalar product and norm).** Let
\[
  f = \begin{pmatrix} 1 & 0 \\ -2 & 2 \end{pmatrix}
\]
and
\[
  g = \begin{pmatrix} 4 & 2 \\ -1 & -3 \end{pmatrix}.
\]
What is the scalar product \( f \cdot g \)? What is the norm \( ||f|| \)?

**Example 3.2.4.** One example of an orthonormal basis for 2×2-matrices are the four basis images below:
\[
  \Phi_{11} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}/2 \quad \Phi_{12} = \begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix}/2
\]
\[
  \Phi_{21} = \begin{bmatrix} 1 & -1 \\ -1 & -1 \end{bmatrix}/2 \quad \Phi_{22} = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}/2
\]
Check that they are orthonormal! Study the image (the matrix)
\[
  f = \begin{bmatrix} 9 & -1 \\ 5 & 7 \end{bmatrix}
\]
What are the coordinates for \( f \) in the new basis? According to Theorem ??? the coordinates for \( f \) is given by the scalar products
\[ x_{ij} = f \cdot \Phi_{ij} = \sum_{\lambda,\mu} f(\lambda, \mu)\Phi_{ij}(\lambda, \mu) \]
For the matrix \( f \) above we get
\[ x_{11} = 10, \quad x_{21} = -2, \quad x_{12} = 4, \quad x_{22} = 6. \]
In other words we have
\[ f = x_{11}\Phi_{11} + x_{21}\Phi_{21} + x_{12}\Phi_{12} + x_{22}\Phi_{22}. \]
or
\[ f_{ij} = x \cdot \Phi_{ij} = \sum_{\lambda,\mu} f(\lambda, \mu)\Phi_{ij}(\lambda, \mu). \]
Check that
\[
  \begin{bmatrix} 9 & -1 \\ 5 & 7 \end{bmatrix} = 10 \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}/2 - 2 \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}/2 + 4 \begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix}/2 + 6 \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}/2
\]
Sometimes we collect the new coordinates in a matrix
\[
  x = \begin{bmatrix} 10 & 4 \\ -2 & 6 \end{bmatrix}.
\]
Example 3.2.5. walsh One example of an orthonormal basis consists of the Walsh matrices $\Phi_{ij}$, named after Joseph L. Walsh, who proposed such structures in 1923. For $4 \times 4$-matrices the first nine are given by

$$
\Phi_{11} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}, \quad \Phi_{12} = \begin{bmatrix} -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}, \quad \Phi_{21} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ -1 & -1 \\ -1 & -1 \end{bmatrix}, \quad \Phi_{13} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 \end{bmatrix}, \quad \Phi_{22} = \begin{bmatrix} -1 & -1 & 1 & 1 \\ -1 & -1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \end{bmatrix}, \quad \Phi_{31} = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}, \quad \Phi_{14} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 \end{bmatrix}, \quad \Phi_{23} = \begin{bmatrix} -1 & -1 & 1 & 1 \\ -1 & -1 & 1 & 1 \\ -1 & -1 & 1 & 1 \\ 1 & 1 & -1 & -1 \end{bmatrix}, \quad \Phi_{32} = \begin{bmatrix} -1 \\ -1 \\ 1 \\ 1 \end{bmatrix}, \quad \Phi_{33} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ -1 & -1 \\ -1 & -1 \end{bmatrix}, \quad \Phi_{34} = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}
$$

Check that they are orthogonal! The norm of the matrices are 4. According to Theorem ?? the coordinates for $f$ in a Walsh basis is given by the scalar products

$$
x_{ij} = f \cdot \Phi_{ij} / 16 = \frac{1}{16} \sum_{\lambda,\mu} f(\lambda,\mu) \Phi_{ij}(\lambda,\mu)
$$

If $x_{00}$, $x_{01}$ and $x_{10}$ are calculated as above, according to theorem ?? the best approximation (in a least squares sense) of an image $f$ using images that are a linear combination of $\Phi_{00}$, $\Phi_{01}$ and $\Phi_{10}$ is given by

$$
x_{00}\Phi_{00} + x_{01}\Phi_{01} + x_{10}\Phi_{10}
$$

### 3.3 Fourier transform

The second part of the lecture is on the Fourier transform. The discrete fourier transform is also described in Chapter 3.4 in Szelisky.

#### The Discrete Fourier Transform (DFT) in one variable

Let 

$$
f = \begin{bmatrix} f(1) \\ \vdots \\ f(N) \end{bmatrix}
$$

be a vector in $\mathbb{C}^N$.

**Definition 3.3.1.** *The Discrete Fourier Transform (DFT) of the $\mathbb{C}^N$-vector $f$ is the $\mathbb{C}^N$-vector $F$, whose components

$$
F(u) = \sum_{x=1}^{N} f(x) \exp[-i2\pi(u-1)(x-1)/N], \quad u = 1, \ldots, N.
$$

The equation can be seen as a discretized version of the continuous version of the fouriertransform. For a given $\omega_N$ introduce the complex constant

$$
\omega_N = \exp(-i2\pi/N).
$$
Note that $\omega_N$ lies on the complex unit circle, i.e. $|\omega_N| = 1$. For example, we have $\omega_2 = -1$, $\omega_4 = -i$ and $\omega_8 = (1 - i) / \sqrt{2}$ ($\omega_N$ is one of the $N$:th roots of unity, $\omega_N^N = 1$.) The discrete Fourier transform can then be written

$$F(u) = \sum_{x=1}^{N} f(x) \omega_N^{(x-1)(u-1)}, \quad u = 1, \ldots, N$$

The mapping from a vector $f$ to its Fourier transform $F$ is linear and can be expressed by as $F = Mf$ for some matrix $M$. This matrix, the Fourier matrix is important.

**Definition 3.3.2.** Set $\omega_N = \exp(-i2\pi/N)$. The Fourier matrix $F_N$ is defined as

$$F_N = \begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \omega_N & \omega_N^2 & \ldots & \omega_N^{N-1} \\ 1 & \omega_N^2 & \omega_N^4 & \ldots & \omega_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_N^{N-1} & \omega_N^{2(N-1)} & \ldots & \omega_N^{(N-1)(N-1)} \end{pmatrix} = [\omega_N^{(i-1)(j-1)}]_{i,j=1,\ldots,N}.$$  

The DFT can then be written as a matrix multiplication

$$f \rightarrow F = F_N f.$$  

The discrete Fourier transform is invertible. So it possible for each transform $F$ to calculate $f$ so that $F = \frac{1}{N}Ff$. This is captured in the following theorem.

**Theorem 3.3.1.** For the Fourier matrix the following holds,

$$F\overline{F} = N I.$$  

From this we obtain $F^{-1} = \frac{1}{N} \overline{F}$. The inverse Fourier transform is thus

$$f = \overline{F}F \iff f(x) = \frac{1}{N} \sum_{u=1}^{N} F(u) \omega_N^{(x-1)(u-1)}, \quad x = 1, \ldots, N.$$  

**Proof.** The proof is obtaine by multiplying $F$ with $\overline{F}$, and using the fact that

$$\omega_N \overline{\omega_N} = 1, \quad \sum_{j=1}^{N} (\omega_N^p)^{(j-1)} = \frac{1 - \omega_N^{Np}}{1 - \omega_N} = 0$$

for $p = 1, \ldots, N - 1$. (Geometrisk summa.)

The inversion formula can also be written

$$f(x) = \sum_{x=1}^{N} F(u) \exp[i2\pi(u-1)(x-1)/N], \quad x = 1, \ldots, N \quad (3.3)$$

The equations for the fourier transform (3.2) and its inverse (3.3) are natural, if you think about them as the discretization of corresponding equations for the continuous fourier transform. Nevertheless the inversion equation (3.3) has to be proven; that it is similar to the corresponding equation for the continuous Fourier transform does not constitute a proof.

The Fourier transform is thus an invertible linear transformation $\mathbb{C}^N \rightarrow \mathbb{C}^N$. It can also be used to calculate approximations to the continuous Fourier transform.

Using the definition, the Fourier transform can be calculated using $N^2$ multiplications (and as many additions). The following theorem can be used to substantially speed up calculations.
**Theorem 3.3.2.** (The Fast Fourier Transform FFT) The Fourier matrix can be factorized as:

\[ F_{2N} = \left[ \begin{array}{cc} I & D_N \\ I & -D_N \end{array} \right] \left[ \begin{array}{cc} F_N & 0 \\ 0 & F_N \end{array} \right] P_N, \]

where

\[ D_N = \text{diag}(1, \omega_{2N}, \omega_{2N}^2, \ldots, \omega_{2N}^{N-1}) = \begin{bmatrix} 1 & 0 & 0 & \ldots & 0 \\ 0 & \omega_{2N} & 0 & \ldots & 0 \\ 0 & 0 & \omega_{2N}^2 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & \omega_{2N}^{N-1} \end{bmatrix} \]

and \( P_N \) are permutation matrices of order \( 2N \times 2N \) that maps \((x(1), x(2), \ldots, x(2N)) \rightarrow (x(1), x(3), \ldots, x(2N - 1), x(2), x(4), \ldots, x(2N))\).

**Proof.** The main idea for the proof is easiest to see for \( N = 2 \). The proof for the general case is similar, but needs a bit more writing. For \( N = 2 \) write \( \omega = \omega_{2N} = \omega_4, \tau = \omega^2 = \omega_N \). Then we have

\[ F_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 \\ 1 & \omega^4 & \omega^6 & \omega^8 \\ 1 & \omega^3 & \omega^6 & \omega^9 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega^2 & \omega^3 & \omega^4 \\ 1 & \omega^6 & \omega^9 & \omega^{12} \\ 1 & \omega^3 & \omega^6 & \omega^9 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \]

Using this theorem, the calculation of the DFT for order \( 2^{n+1} \) can be done by two calculations of the DFT of order \( 2^n \), which in turn can be calculated by four DFT’s of order \( 2^{n-1} \) etc. If \( \mu_n \) is the number of multiplications needed for calculating DFT of order \( 2^n \) then we have

\[ \mu_n = 2\mu_{n-1} + 2^{n-1}. \]

This is a recursion formula. Using \( \mu_1 = 0 \), we get

\[ \mu_n = \frac{n2^n}{2} = \frac{N \log_2 N}{2} \quad \text{for} \quad N = 2^n. \]

If we study e.g. \( N = 1024 \), then the number of multiplications needed for the calculation of DFT according to the definition is \( N^2 \approx 10^6 \), whereas the number of multiplications for the FFT is \( N \log_2 N/2 \approx 10^3 \cdot 10/2 \approx 10^4 \). For this case the FFT is a factor of 100 faster!

**The DFT in two variables**

As we discussed earlier in this lecture, the set of \( M \times N \) matrices can be thought of as a linear vector space. The DFT of an image is a linear mapping. After column stacking of an images \( f \), the mapping can be written

\[ \tilde{g} = A\tilde{f} \]
Multiplication with the original matrix $f$ might not be meaningful. There are, however, certain linear mappings that can be written in terms of the original matrix $f$. A linear mapping $f \rightarrow g$ is said to be separable if it can be written as a product

$$g = \Phi f \Psi,$$

for a matrix pair $(\Phi, \Psi)$.

Here we will introduce the DFT for images and show that it is separable. First we define the two-variable Discrete Fourier Transform (DFT). Analogous to (3.2) this is defined as

$$F(u, v) = \sum_{x=1}^{M} \sum_{y=1}^{N} f(x, y) e^{-i2\pi((u-1)(x-1)/M+(v-1)(y-1)/N)},$$

where

$$f(x, y) = \frac{1}{MN} \sum_{u=1}^{M} \sum_{v=1}^{N} F(u, v) e^{i2\pi((u-1)(x-1)/M+(v-1)(y-1)/N)},$$

This can be written

$$F(u, v) = \frac{1}{MN} \sum_{x=1}^{M} \sum_{y=1}^{N} f(x, y) \exp(-i2\pi((u-1)(x-1)/M+(v-1)(y-1)/N)),$$

which proves the following theorem.

**Theorem 3.3.3.** The two-dimensional Discrete Fourier Transform (DFT) is given by

$$f \rightarrow F = \frac{1}{MN} \mathcal{F}_M f \mathcal{F}_N \quad (3.4)$$

The inverse Fourier transform is

$$F \rightarrow \mathcal{F}_M f \mathcal{F}_N \quad (3.5)$$

These equations show that FFT can be used for images as well. This makes the calculation of the fourier transform efficient. One application is within convolution (as we will discuss in the next lecture).

For those who can read swedish, Gunnar Sparrs original text can be found at [http://www.ctr.maths.lu.se/media/FMAN20/2016/dftmm.pdf](http://www.ctr.maths.lu.se/media/FMAN20/2016/dftmm.pdf)

The text was written for the image analysis course and provides this overview on linear algebra, convolution and Fourier transform.

**Example 3.3.1.** The fourier transform $F$ of an image $f$, can be viewed as a change of coordinates. Can we get a feeling for what the basis elements look like, what the fourier transform means. In this example we are going to study $32 \times 32$ images. We will use the $\chi_{ij}$ to denote the natural (canonical) basis for $32 \times 32$-matrices consisting of matrices

$$E_{ij} = \begin{bmatrix}
0 & \cdots & 0 & \cdots & 0 \\
0 & \cdots & 1 & \cdots & 0 \\
\cdots & \cdots & \cdots & \cdots & \cdots \\
0 & \cdots & 0 & \cdots & 0
\end{bmatrix},$$

which has zeros everywhere except at position $i, j$, where there is a single one.
Figure 3.1: The right image is the Fourier transform of the left. The right image $E_{43}$ is chosen so that it is zero everywhere except at position row 4 and column 3, where there is a one. The left image (the inverse Fourier transform of $E_{43}$) is complex. The real part is shown as a gray-scale image. Notice that there are $(4 - 1)$ periods along the y-axis and $(3 - 1)$ periods along the x-axis in the sinusoidal image to the left.

Let's study the inverse Fourier transform of one such image, e.g. $F = E_{43}$. In figure 3.1, the real part of the inverse Fourier transform of $f = E_{43}$ is shown to the left and $E_{43}$ is shown to the right. Notice that the left image is sinusoidal. Notice also that there are 3 periods on the y-axis and 2 periods on the x-axis. In fact the inverse Fourier transform of $E_{ij}$ has $(i - 1)$ periods on the y-axis and $(j - 1)$ periods on the x-axis. Thus the coordinate $F_{43}$ specifies how much of the sinusoidal pattern to the left in figure 3.1 to use.
Chapter 4

Lecture 4: Convolution, Filtering, Convolution theorem

4.1 Introduction

This lecture is about convolution and its connection to the Fourier transform. **Convolution:** Every linear operator on images, which is translation invariant can be represented as a convolution. Read chapter 3.2-3.3 in Szelisky. **Convolution theorem:** The convolution theorem states that if $f = g * h$ then

$$F(u, v) = G(u, v)H(u, v).$$

In other words if the image $f$ is the result of convolving $g$ with $h$, then the Fourier transform of $F$ is the elementwise product of the Fourier transforms of $g$ and $h$. This ties together the concepts of convolution and the Fourier transform.

4.2 Convolution

One and two-dimensional convolution is defined and describes in chapter 3.2 in Szelisky. The convolution is an operation that takes two images $f$ and $h$ as input and produces a new image $g = f * h$. For simplicity here we are going to assume that images are infinite, but that the non-zero elements are at a finite number of positions. For an $M \times N$ image, we can think of the image being extended with zeroes outside the actual image.

**Definition 4.2.1.** The convolution is an operation that takes two images $f$ and $h$ as input and produces a new image $g = f * h$. It is defined as

$$g(i, j) = \sum_u \sum_v f(i - u, j - v)h(u, v), \quad (i, j) \in \mathbb{Z}^2. \quad (4.1)$$

To understand what convolution means it is sometimes useful to flip the second operand, $h$, i.e. introduce

$$\tilde{h}(u, v) = h(-u, -v).$$

The convolution can then be re-written as

$$g(i, j) = \sum_u \sum_v f(i - u, j - v)\tilde{h}(-u, -v), \quad (i, j) \in \mathbb{Z}^2. \quad (4.2)$$

Change summation variable $y = -u$ and $x = -v$,

$$g(i, j) = \sum_y \sum_x f(i + y, j + x)\tilde{h}(y, x), \quad (i, j) \in \mathbb{Z}^2. \quad (4.3)$$
This operation is called cross-correlation or sliding dot product. Study the result at position (0, 0), i.e.

\[ g(0, 0) = \sum_y \sum_x f(y, x) h(y, x) \]  

(4.4)

and notice that this is simply the scalar product between the image \( f \) and \( \hat{h} \). Study the result at another position, for example (7, 3), i.e.

\[ g(7, 3) = \sum_y \sum_x f(7 + y, 3 + x) \hat{h}(y, x) \]  

(4.5)

and notice that this is simply the scalar product between a translated version of the image \( f \) and \( \hat{h} \).

The cross-correlation is sometimes easier to understand. What are the arguments for using the convolution operator, which involves flipping the operand? One strong argument for using convolution is that it several nice properties, such as

\[
\begin{align*}
  f * h &= h * f, \\
  f * (g * h) &= (f * g) * h, \\
  f * (g + h) &= f * g + f * h, \\
  a(f * g) &= (af) * g, \\
  \delta * f &= f, \\
  \partial(f * g) &= (\partial f) * g,
\end{align*}
\]

**Example 4.2.1.** Often it is useful to think of \( f \) as a large image and \( h \hat{a} \) as a small image in the sense that \( h \) is non-zero only at a few positions. Let \( h \) be such that

\[ h(0, 0) = 2, \ h(1, 0) = 1, \ h(-1, 0) = 1, \ h(0, 1) = 1, \ h(0, -1) = 1 \]

and \( h \) zero everywhere else. Let \( f \) be the image

\[
 f = \begin{bmatrix}
 10 & 0 & 10 & 10 & 10 \\
 0 & 0 & 10 & 10 & 10 \\
 0 & 0 & 10 & 20 & 10 \\
 0 & 0 & 10 & 10 & 10 \\
 0 & 10 & 10 & 10 & 10 \\
\end{bmatrix}
\]

Note that we here assume that \( f \) is indexed so that the upper left corner has indices (1, 1). We also assume that \( f \) is zero everywhere else. So for example \( f(-2, 1) = 0 \) and \( f(0, 0) = 0 \).

The convolution is found by sliding the flipped version of \( h \) over \( f \) and computing the scalar product at each place. Think of

\[
 \hat{h} = \begin{bmatrix}
 0 & 1 & 0 \\
 1 & 2 & 1 \\
 0 & 1 & 0 \\
\end{bmatrix}
\]

as a small window sliding over \( f \), where the (0, 0)-element of \( \hat{h} \) is considered the centre point. To compute \( g(1, 1) \) we place the centre point of \( \hat{h} \) over the \( f(1, 1) \) element and cut out corresponding sub-matrix centered around \( f(1, 1) \), i.e.

\[
\begin{bmatrix}
 0 & 0 & 0 \\
 10 & 0 & 10 \\
 0 & 0 & 0 \\
\end{bmatrix}
\]

and calculate the scalar product. The result is 20. To calculate \( g(1, 2) \) we cut out the sub-matrix one step to the right, i.e.

\[
\begin{bmatrix}
 0 & 0 & 0 \\
 10 & 0 & 10 \\
 0 & 0 & 10 \\
\end{bmatrix}
\]
and calculate the scalar product with $\mathbf{h}$. The result is 20 again. To calculate $g(1, 3)$ we cut out the sub-matrix around the element $(1, 3)$, i.e.

$$
\begin{bmatrix}
0 & 0 & 0 \\
0 & 10 & 10 \\
0 & 10 & 10 \\
\end{bmatrix}
$$

The result this time is 40. The result $g(i, j)$ for $i = 1, \ldots, 5, j = 1, \ldots, 5$ is

$$
\begin{bmatrix}
20 & 20 & 40 & 50 & 40 \\
10 & 10 & 50 & 70 & 50 \\
0 & 10 & 60 & 80 & 60 \\
0 & 20 & 50 & 70 & 50 \\
10 & 30 & 50 & 50 & 40 \\
\end{bmatrix}
$$

For images taken with an ordinary camera, it is somewhat strange to assume that all pixels outside the field of view are 0. Sometimes we are only interested in the result for which we haven’t used any information outside the field of view. For this example these are the values $g(i, j)$ for $i = 2, \ldots, 4, j = 2, \ldots, 4$, i.e.

$$
\begin{bmatrix}
* & * & * & * & * \\
* & 10 & 50 & 70 & * \\
* & 10 & 60 & 80 & * \\
* & 20 & 50 & 70 & * \\
* & * & * & * & * \\
\end{bmatrix}
$$

In other situations it is relevant to calculate all non-zero values of $g$. In our case this would mean calculating $g(i, j)$ for $i = 0, \ldots, 6, j = 0, \ldots, 6$, i.e.

$$
\begin{bmatrix}
0 & 10 & 0 & 10 & 10 & 10 & 0 \\
10 & 20 & 20 & 40 & 50 & 40 & 10 \\
0 & 10 & 10 & 50 & 70 & 50 & 10 \\
0 & 0 & 10 & 60 & 80 & 60 & 10 \\
0 & 0 & 20 & 50 & 70 & 50 & 10 \\
0 & 10 & 30 & 50 & 50 & 40 & 10 \\
0 & 0 & 10 & 10 & 10 & 10 & 0 \\
\end{bmatrix}
$$

These three different versions of convolution are found in many programming languages, e.g. matlab, python, julia. In matlab they are called

- `conv2(f,h,'same')` % produces a result $g$ of the same size as $f$
- `conv2(f,h,'valid')` % does not use any pixels outside the field of view
- `conv2(f,h)` % calculates all non-zero values of $g$. Size of $g$ is larger.

If the image is large and if $h$ is small, the size difference is relatively small. There is also a fourth version on how to handle the border effects. That is to consider the image to be double periodic. In other words we think of the pixels to the right of the right edge of the image, to be that of the left part of the image. This fourth version of the convolution is used in the next sections. Using periodic convolution we can prove a nice theorem that links convolution and the fourier transform.

### 4.2.1 Periodic convolution in one dimension

This is a short description of the periodic convolution and the convolution theorem.

Given a column vector of length $N$,

$$
f = \begin{bmatrix}
f(1) \\
\vdots \\
f(N)
\end{bmatrix}
$$
Think of this as representing a periodic extension with period $N$. We will in the sequel use the same variable $f$ for both the original vector of length $N$ and the periodic extension. In this sense we have e.g. $f(0) = f(N)$, and

$$f = \begin{bmatrix}
  \vdots \\
  f(N - 1) \\
  f(N) \\
  f(1) \\
  \vdots \\
  f(N) \\
  f(0) \\
  f(1) \\
  \vdots 
\end{bmatrix}.$$ 

$N$-periodic sequences are characterized by

$$f(x + N) = f(x) \quad \text{for all} \quad x.$$ 

If also $h$ is a $N$-periodic sequence, then the following operator is defined.

**Definition 4.2.2.** The periodic convolution of $f$ with $h$ is the periodic sequence, whose element at position $x$ is

$$g(x) = f * h(x) = \sum_{m=1}^{N} h(x - m + 1)f(m), \quad x \in \mathbb{Z}.$$ 

In order to calculate $g(x)$ for every $x = 1, \ldots, N$ requires $\approx N^2$ multiplications. It turns out that FFT can be used to significantly reduce this number.

To see the structure of the $g$-vector.

$$g(N) = \sum_{m=1}^{N} h(N - m)f(m)$$

$$g(N + 1) = \sum_{m=1}^{N} h(N + 1 - m)f(m)$$

$$\vdots$$

$$g(2N - 1) = \sum_{m=1}^{N} h(2N - 1 - m)f(m)$$

Because of the periodic nature, i.e. $g(N) = g(0)$, $g(N + 1) = g(1)$ \ldots, this can be written

$$\begin{bmatrix}
  g(1) \\
  g(2) \\
  \vdots \\
  g(N)
\end{bmatrix} = \begin{bmatrix}
  h(1) & h(N) & \ldots & h(3) & h(2) \\
  h(2) & h(1) & \ddots & \ddots & \ddots \\
  \vdots & \ddots & \ddots & \ddots & \ddots \\
  h(N) & h(N - 1) & \ldots & h(2) & h(1)
\end{bmatrix} \begin{bmatrix}
  f(1) \\
  f(2) \\
  \vdots \\
  f(N)
\end{bmatrix}$$ 

(4.7)

**Definition 4.2.3.** The matrix in (4.7) is said to be the circulant matrix of $h$.

Thus we have shown that

$$g = h * f \iff g = C_h f.$$ 

(4.8)

**Remark.** Note the band structure of the circulant matrix. There is a band with the same value on the diagonals. Note also that when such an diagonal hits the last row it continues on the first row. A circulant matrix has many interesting properties. One of the most remarkable ones is that they all have the same set of eigenvectors.
Lemma 4.2.1. Every circulant matrix has the same set of eigenvectors, i.e. the columns \( \phi_i \) of the conjugated Fourier matrix \( \mathcal{F}_N \). The eigenvalue \( \lambda_i \) corresponding to eigenvector \( \phi_i \) is the \( i \):th Fourier coefficient of \( h \), i.e.

\[
\lambda_i = H(i) = \sum_{j=1}^{N} h(j) \omega^{ij}.
\]

Proof. The equation (4.8) gives \( C_h \phi_i = \phi_i * h \). We calculate the element \( x \) of the convolution \( \phi_i * h \), i.e.

\[
\sum_{k} \omega^{-(x-k)i} h(k) = \omega^{-xi} \sum_{k} \omega^{ki} h(k) = H(i) \omega^{-xi}.
\]

This is element nr \( x \) in the vector \( \phi_i \), multiplied with \( H(i) \), which proves the lemma.

The columns of \( \mathcal{F}_N \) are linearly independent. Therefore \( C_h \) has a basis of eigenvectors and thus \( C_h \) is diagonalizable and:

\[
C_h \mathcal{F}_N = \mathcal{F}_N \Lambda_H \iff \mathcal{F}_N C_h = \Lambda_H \mathcal{F}_N \iff \mathcal{F}_N^{-1} C_h \mathcal{F}_N = \lambda H,
\]

with

\[
\Lambda_H = \text{diag}(H(1), \ldots, H(N)).
\]

In rewriting (4.9) we used the fact that \( \mathcal{F}_N \mathcal{F}_N^* = NI \).

From this useful lemma follows the convolution theorem,

Theorem 4.2.1.

\[
\mathcal{F}_N h * f = \begin{bmatrix} H(1) F(1) \\ \vdots \\ H(N) F(N) \end{bmatrix}.
\]

Proof. According to (4.8) and (4.9) follows

\[
\mathcal{F}(h * f) = \mathcal{F}_N C_h f = \Lambda_H \mathcal{F}_N f = \Lambda_H F = \begin{bmatrix} H(1) F(1) \cdots H(N) F(N) \end{bmatrix}^T.
\]

The convolution theorem can be used to efficiently calculate the convolution \( g = f \ast h \) according to the following scheme:

1. Calculate the fourier transforms \( F \) and \( H \).
2. Then generate \( G \) by elementwise multiplication, i.e. \( G(u) = F(u) H(h) \).
3. The result \( g \) is obtained from \( G \) using (3.5).

This involves three FFT operations. The number of multiplications needed are

\[
2 \cdot \frac{N \log N}{2} + N + N + \frac{N \log N}{2} \approx \frac{3}{2} N \log N.
\]

For large \( N \) this is substantially less than \( \approx N^2 \), which is the number of multiplications needed using the original implementation.
4.2.2 Convolutions for images

Let \( f \) be a \( M \times N \)-matrix. A periodic extension of a matrix is obtained by \( N \)-periodic extension of every row and then by \( M \)-periodic extension of every column. In this way we obtain a double periodic infinite matrix.

Similar to before we will use the same notation \( f \) for the (double)-periodic extension, which has the property that

\[
    f(x + jM, y + kN) = f(x, y) \quad \text{for every} \quad x, y \in \mathbb{Z} \quad \text{and every} \quad j, k \in \mathbb{Z}.
\]

The convolution of such periodic matrices is defined as

**Definition 4.2.4.**

\[
    g(x, y) = f * h(x, y) = \sum_{m=1}^{M} \sum_{n=1}^{N} f(m, n)h(x - m, y - n).
\]

\( f * h \) is periodic in the same way as \( f \) and \( h \). Similar to the one-dimensional case we will show that convolutions can be expressed using circulant matrices. We have

\[
    g(x, y) = \sum_{n} \left( \sum_{m} f(m, n)h(x - m, y - n) \right).
\]

The inner sum is

\[
    \gamma_n(x, y) = \sum_{m} f(m, n)h(x - m, y - n).
\]

For a fixed \( x, n \) this is a convolution \( x - m \) in \( h \) with column \( n \) of \( f \). By stacking the result \( y \) and using (4.8) we find that

\[
    \gamma_n^T(\cdot, n) = C_{h(\cdot, x-n)}f^T(\cdot, n).
\]

Here \( C_{h(\cdot, x-n)} \) is the circulant matrix for the \((y-n)\)th column of \( h \). Inserting this in (4.10) gives

\[
    g(\cdot, y) = \sum_{n} C_{h(\cdot, y-n)}f(\cdot, n).
\]

Using matrix notation this can be written

\[
    g(\cdot, y) = \left[ C_{h(\cdot, y-n)} C_{h(\cdot, y-N)} \cdots C_{h(\cdot, y+1)} \right] \begin{bmatrix} f(\cdot, 1) \\ f(\cdot, 2) \\ \vdots \\ f(\cdot, N) \end{bmatrix}.
\]

The column vector to the right is simply the result \( \tilde{f} \) of column stacking the image \( f \). Through stacking such equations for \( y = 1, \ldots, N \) we obtain a result

\[
    \tilde{g} = C_{h(\cdot, \cdot)}\tilde{f}
\]

where the coefficient matrix is called the block circulant matrix.

**Definition 4.2.5.** The block circulant matrix of the \( M \times N \)-matrix \( f \) is the matrix of type \( MN \times MN \)

\[
    C_{h(\cdot, \cdot)} = \begin{bmatrix}
        C_{h(\cdot, 1)} & C_{h(\cdot, N)} & \cdots & C_{h(\cdot, 3)} & C_{h(\cdot, 2)} \\
        C_{h(\cdot, 2)} & C_{h(\cdot, 1)} & \cdots & \cdots & \cdots \\
        \vdots & \cdots & \ddots & \ddots & \cdots \\
        \vdots & \ddots & \ddots & \ddots & \ddots \\
        C_{h(\cdot, N)} & C_{h(\cdot, N-1)} & \cdots & C_{h(\cdot, 2)} & C_{h(\cdot, 1)}
    \end{bmatrix}
\]

where the blocks are circulant matrices for the columns of \( h \).
Similar to the one-dimensional case we will show that this can be used to calculate convolutions efficiently using the FFT.

**Lemma 4.2.2.** All block circulant matrices has the same set of eigen vectors

\[
\Phi_{k,j} = \begin{bmatrix}
\omega^{-1k} \phi_j \\
\omega^{-2k} \phi_j \\
\vdots \\
\omega^{-Mk} \phi_j
\end{bmatrix}, \quad j = 1, \ldots, N, \quad k = 1, \ldots, M,
\]

where \( \phi_j \) is the \( j \):th column of \( \mathcal{F}_N \). The eigenvalue to \( C_{h(\cdot, \cdot)} \) corresponding to the eigenvector \( \Phi_{k,j} \) is \( H(k, j) \), where \( H \) is the discrete Fourier transform of \( h \). In other words

\[
C_{h(\cdot, \cdot)} \Phi_{k,j} = H(k, j) \Phi_{k,j}.
\]

**Proof.** Introduce a notation for the "partial Fouriertransform" of \( h \) with respect to the first variable,

\[
p_h(i, y) = \sum_{x=1}^N h(x, y) \omega^{(x-1)(i-1)}.
\]

Partial fourier transform with respect to the second variable then gives

\[
H(k, j) = \sum_{y=1}^N p_h(i, y) \omega^{(y-1)(k-1)}.
\]

(4.12)

Study the equation (4.11) for the part of \( \tilde{g} \) corresponding to the \( y \):th column of \( g \). For this we have

\[
g(\cdot, y) = \sum_{\nu=1}^M C_{(\cdot, y+M-\nu)} \omega^{-(\nu-1)(k-1)} \phi_j,
\]

where we have used the periodicity. According to lemma 4.2.1 and (4.12) this can be written

\[
g(\cdot, y) = \sum_{\nu=1}^M \omega^{-(\nu-1)(k-1)} p_h(x + M - \nu, j) \phi_j = \omega^{-(x+M)(k-1)} \sum_{\nu=1}^M \omega^{x+M-(\nu-1)(k-1)} p_h(x + M - \nu, j) \phi_j = H(k, j) \omega^{-(x-1)(k-1)} \phi_j.
\]

This concludes the proof.

In the sequel we will use \( \mathcal{F} = \mathcal{F}_N \), for brevity. Introduce the matrix

\[
\Psi = \begin{bmatrix}
\omega^{-11} \mathcal{F} & \omega^{-12} \mathcal{F} & \ldots & \omega^{-1M} \mathcal{F} \\
\omega^{-21} \mathcal{F} & \omega^{-22} \mathcal{F} & \ldots & \omega^{-2M} \mathcal{F} \\
\vdots & \vdots & \ddots & \vdots \\
\omega^{-M1} \mathcal{F} & \omega^{-M2} \mathcal{F} & \ldots & \omega^{-MM} \mathcal{F}
\end{bmatrix}.
\]

Note the close resemblance to \( \mathcal{F} \). It can be seen as a combination of \( \mathcal{F}_N \) and \( \mathcal{F}_M \), where one matrix is used blockwise multiplied with elements of the other in order to produce the final matrix. Matrices constructed in this way is denoted *Kronecker products*.

Also note that repeated use of Theorem 3.3.1 gives

\[
\Psi \bar{\Psi} = MN I \iff \Psi^{-1} = \frac{1}{MN} \bar{\Psi}.
\]

Thus the matrix \( \Psi \) has the same role for block circulant matrices as \( \mathcal{F} \) has for circulant matrices. From Lemma 4.2.2 follows that the columns of \( \Psi \) are the eigenvectors of \( C_{h(\cdot, \cdot)} \), and

\[
C_{h(\cdot, \cdot)} \Psi = \Psi \Lambda_H \quad \text{där} \quad \Lambda_H = \text{diag}(H(1, 1), H(1, 2), \ldots, H(M, N)).
\]
The convolution \( h \ast f \) can now be calculated. Combine (4.11) with (4.13) to get \( g = h \ast f \),

\[
\Psi \tilde{g} = \Psi C_{h(\cdot,\cdot)} \tilde{f} = \Lambda H \Psi \tilde{f}.
\]

Since \( \Psi \tilde{f} = \tilde{F} \), \( \Psi \tilde{g} = \tilde{G} \), (think!) we have shown the following theorem.

**Theorem 4.2.2.** For the discrete fourier transform \( G \) of \( g = h \ast f \) holds

\[
G(k, j) = H(k, j) F(k, j).
\]

(We are used to these kind of formulas in the continuous version, but again they have to be prooved.) Thanks to this theorem the fast fourier transform can be used to calculate two-dimensional convolutions in the same way as we did for one-dimensional convolutions, by (i) first calculating \( F \) and \( H \) using (3.4), (ii) multiply \( F \) and \( H \) elementwise to produce \( G \), (iii) calculate \( g \) using the inverse fourier transform (3.5). In steps (i) and (iii) we use the fast fourier transforms.
Chapter 5

Lecture 5: Scale space, Features, Edges, Ridges, Blobs

Convolution with a Gaussian functions makes the image a little bit blurrier. Repeated convolutions with a Gaussian function makes the images increasingly blurrier.

5.0.1 Understanding Convolutions

Convolution with a small patch combines local information in an image. Fourier transform can be used to understand the result of convolution. Linear algebra can be used to understand the result of convolution.

Scale-space: Read chapter 3.5 in Szelisky.

Feature detection, points, patches, edges, lines: Read chapter 4 in Szelisky.

Blob detection, sub-pixel: Look at http://www.ctr.maths.lu.se ... localmaxblobcut.html which contains an example of detecting centre-points of blobs in images with sub-pixel precision.

Definition 5.0.1. A filterbank is a set of $K$ convolution kernels $h$. A multi-convolution takes an image $f$ and a filterbank $\mathbf{h}$ as input and produces a tensor of $K$ images $g$ according to

$$g(i,j,k) = \sum_u \sum_v f(i-u, j-v)h(u,v,k), \quad (i,j) \in \mathbb{Z}^2, \quad k = 1, \ldots, K. \quad (5.1)$$

$$g(i,j) = \sum_u \sum_v f(i-u, j-v)h(u,v), \quad (i,j) \in \mathbb{Z}^2. \quad (5.2)$$
Chapter 6

Lecture 6: Texture, Machine Learning 2

Multiple convolutions produce a new image with many channels. What about using machine learning on each pixel in this new ‘image’.
This makes it possible to combine ideas from Lecture 2 (on pixelwise classification) with spatial information from lectures 3-5.
Texture classification.
More on machine learning.
Chapter 7

Lecture 7: Deep Learning

Multiple convolutions on an image produce a new image with many channels. Machine learning to produce classification in this image.

How about repeating this process. Use multiple convolutions and machine learning to produce a new image.
Chapter 8

Lecture 8: Systems, Testing, Segmentation
Chapter 9

Lecture 9: Fitting, Parameter Estimation
Chapter 10

Lecture 11: Semantic Segmentation, Graph Cuts
Chapter 11

Lecture 12: Review, Wrap Up
Chapter 12

Guest Lecture A: Computer Vision
Chapter 13

Guest Lecture B: Spatial Statistics
Chapter 14

Guest Lecture C: Medical Image Analysis
Chapter 15

Guest Lecture D: Multi-Spectral Imaging
Chapter 16

Guest Lecture E: Ethics