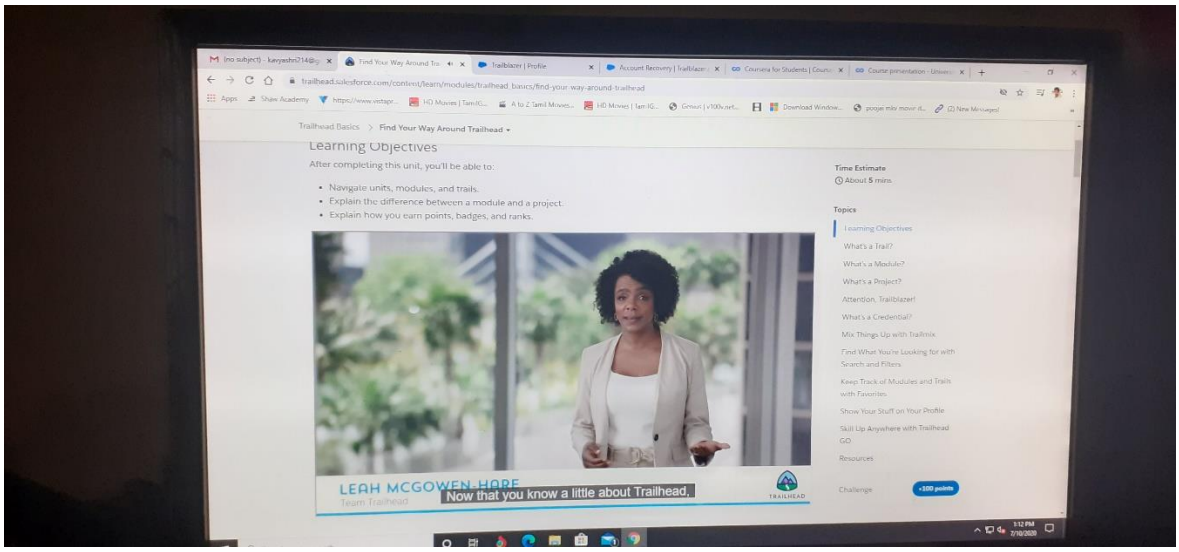
DAILY ASSESSMENT

Date:	**10-7-2020**	**Name:**	**Kavyashree m**
Course:	**salesforce**	**USN:**	**4al15ec036**
Topic:	**Trailhead basics**	**Semester & Section:**	**8th A**
Github Repository:	**kavya**		

FORENOON SESSION DETAILS



Welcome to Trailhead

Every journey starts with a single step—and so does every trail. You've already taken the first step of your learning journey by opening Trailhead.

Trailhead is the fun way to learn. Whether you're an admin just starting out, a graduate fresh from college, a Salesforce user, or someone who loves to learn, there's something on Trailhead for you.

What's a Hands-on Challenge, Anyway?

Learning is the bread and butter of Trailhead, and one of the best ways to learn is by doing. Earning a badge on Trailhead is more than just reading through modules. You also have to complete either a multiple-choice quiz or a hands-on challenge at the end of each

unit. You're probably familiar with multiple-choice quizzes (think Who Wants To Be A Millionaire?), but hands-on challenges are unique to Trailhead.

A hands-on challenge is more involved than a quiz and, as a result, earns you more points. To complete a hands-on challenge, you have to look at a set of requirements and do something in a Salesforce org to meet those requirements. What you have to do depends on what you're learning. You could be writing an Apex class, creating a Lightning web component, or writing a field-level validation formula.

We even provide you with your own org, called a Trailhead Playground, that you can use to solve hands-on challenges.

What's a Trailhead Playground?

When you encounter your first hands-on challenge, you'll see a dropdown that lets you launch your hands-on org. A Trailhead Playground is an org that you can use for hands-on challenges, learning new features, and testing customizations. If you haven't created one, don't worry. You get one automatically when you create a Trailhead account.

What's a Trail?

The essence of Trailhead is, you guessed it, trails. A trail is an ordered group of modules, projects, or both that provides a guided learning path to learning a new skill, product, or role. Because there are so many things you can learn on Trailhead, there are lots of trails currently more than 100 covering a wide range of topics.

What's a Module?

A module covers a single learning topic. While trails are great for learning a comprehensive set of related topics, you can also mix and match modules freely. Say that you're a Salesforce developer with five years of experience, and your org is about to begin the move to Lightning Experience. It's not worth your time to work through the entire Developer Intermediate or even Develop for Lightning Experience trail—you know most of that stuff already. Instead, you can pick and choose the modules that

address the gaps in your knowledge, such as Lightning Web Components for Aura Developers and Lightning Flow.

What's a Credential?

Trailhead is your portal for learning more about Salesforce credentials. There are currently two types of credential: superbadges and Salesforce certifications.

You earn a certification by taking a proctored exam that tests your skills. Salesforce certifications are world-class credentials that demonstrate your abilities to work with Salesforce technology. If you're already certified, Trailhead is where you earn certification maintenance badges, which ensure that you're up to date on the newest Salesforce features.

The credentials tab gives you more information about the different types of Salesforce certifications, including information about exams, links to sign up for instructor-led trainings, and details about what it means to be a Salesforce professional, such as an admin, developer, or consultant.

What If I Want to Skill Up People at My Own Company with Trailhead?

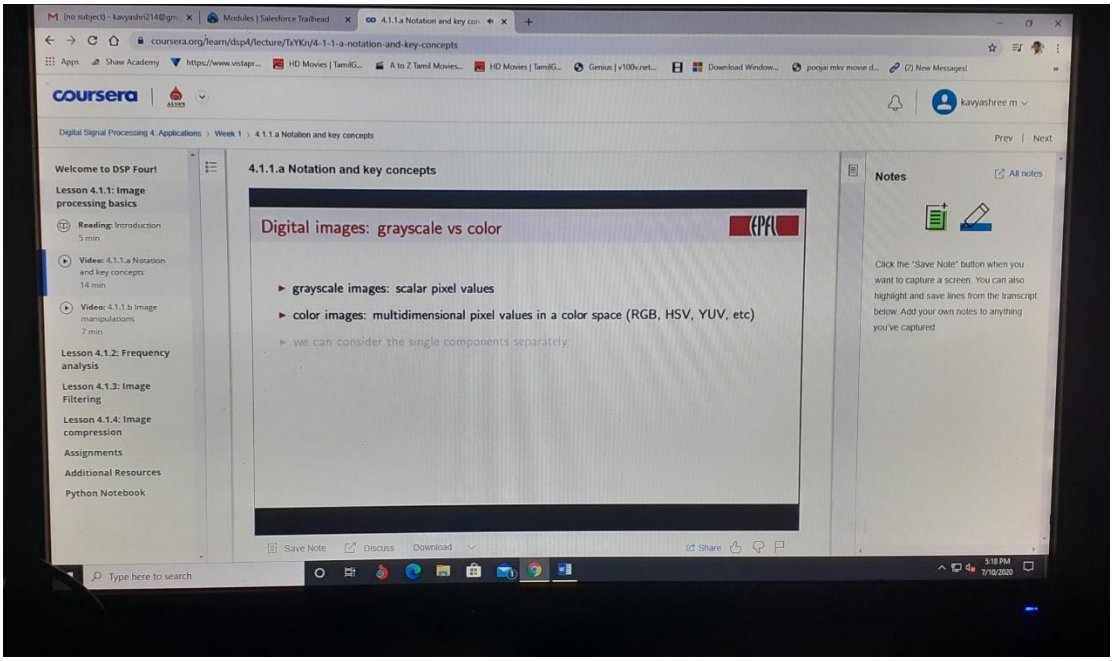
If you love what you see on Trailhead and have already started thinking about how helpful it would be to have your own Trailhead content, we've got you covered with myTrailhead. Just like Salesforce Trailhead, myTrailhead has modules, which each focus on a particular skill and include a series of units. And just like on Salesforce Trailhead, each unit ends with a quiz, and completing a module earns users points, a badge, and a cascade of digital confetti. The difference is that myTrailhead is accessible only to other users at your company, and it includes your company's custom content and branding.

If you're interested in myTrailhead, click For Companies in the navigation banner at the top of the page, then myTrailhead. From here you can see myTrailhead features and get more information.

AFTERNOON SESSION DETAILS

Date:	**10-7-2020**	**Name:**	**Kavyashree m**
Course:	**coursesa**	**USN:**	**4al15ec036**
Topic:	**Image processing basics**	**Semester & Section:**	**8th A**
Github Repository:	**kavya**		

Image of session



Notation and key concepts

In this module, we will look at images as a particular class of multidimensional digital signals. We'll look at different ways to represent this two dimensional signals. And we will explore some basic instances of images and operators that we can apply to them. Before we start, let's make the acquaintance of this cute little

dog. We decided to boost the ratings of our class by using the picture of a puppy, for all the image examples that we'll use in the following.

But even behind the puppy, there's a hard mathematical reality. And so digital images can be expressed as a two-dimensional signal. x[n1, n2] where n1 and n2 are integer numbers, integer indices. And each combination of n1 and n2 indicates a point on a grid. We know already from everyday life that we call these points pixels, as picture elements.Usually the grid is regularly spaced. So we have regular arrangement of points on the grid. And the value of the signal at coordinates n1 and

n2 refers to the pixel's appearance.Take for instance, a grayscale image, a black and white picture.

In digital format, we can encode the appearance of each pixel in the image by associating the grayscale level to a scalar value. The appearance will vary between a lowest value, which we associate to black,

to a highest possible value which we associate to white and all gray levels in between.

For color images, we need something a little bit more complicated. Now the theory of color spaces is a very fascinating subject. But it's way too complicated for even a cursory introduction here. You are all familiar with the RGB color model for instance, which is used in monitors for your computer. The RGB coding associates to each color pixel three scalar values. R, G, and B which represent the amount of red, green, and blue, that we need to mixed up to obtain the desired color. It's a fascinating fact about the human visual system that we can actually represent such a wider array of colors using just three components.

Now when we used a color model it means that each pixel has a multidimensional value. But since these values are independent, we can split the original color images into three fundamental components

that are associated to the components of the vector space. So in the case of RGB for instance, we will have three independent components that encode the red part, the green

part and the blue part. Each of these images is now a scalar image. And so in the following, we will be simply concentrating on these color images for processing. So in image processing, we're moving from one to two dimensions. And we know quite a bit about one-dimensional signal processing already.When we move to two dimensions, something still works.

We can still use some concepts that we used in one dimension. Something unfortunately breaks down and new things appear. So let's look, in turn, at these three scenarios.

The things that work and work rather well, are the concepts of linearity and convolution. The Fourier transform in two dimensions is a simple extension of the one dimensional Fourier transform.

And interpolation and sampling work exactly in the same way in two dimensions. What works less well in image processing, is that, for instance, Fourier analysis, which algorithmically, is just an extension of the 1Dks, becomes much less relevant in the case of images. Filter design is much harder as well, and IIR filters are rare. And later operators are only mildly useful, and the reason is images are very diverse signals.

Imagine a photograph of a landscape where you have all sorts of objects and textures.

Now a linear operator, and in 2D case it would be a linear space invariant operator, would apply the same

kind of transformation to all different parts of an image. Regardless of what they represent. And of course it's kind of difficult to imaging that the same filter, unless it's a very, very simple operation, will yield the desired result when applied to very heterogeneous parts of an image. New concepts that appear in image processing is the fact that we can introduce a new class of manipulations called affine transforms.

These include rotation, scaling, skewing of images. Things you do in Photoshop all the time, the fact that images are finite support signals. By definition you take an image with a camera, and the CCD, the sensor of a camera has a finite surface, so they're intrinsically

finite support. And because of that, an image signal is available in its entirety from the beginning of processing. So while in 1D you can imagine the system that works online with the samples coming and you never know when the samples are going to stop. When it comes to images, you sort of assume that you have the whole image already in memory before you start processing. So causality is less of an issue in image processing. However, all this applies to images not to general 2D signals. Images are a very specialized signal. And they are designed for a very specific type of receiver, the human visual system. So images are a very small subset of 2D signals. And a subset that is imbued with semantics. Now semantics are very very hard to deal with in our linear space and variant paradigm.

Let's now look in more detail at the extension of our digital signal processing paradigm to two dimensions.

So a discrete space signal is a signal that we indicate with this notion, x[n1,n2]. n1 and n2 are two discrete valued indices. The signal could be complex valued. But of course in the case of images, scalar images,

the values are going to be real.

So how do we represent this 2D signal?

Well, from a standard mathematical point of view, we could represent it with a Cartesian plot where we have one axis that indicates the first index. The second axis indicates the second index and the value of the signal is represented as a third coordinate. so we have a 3D plot where the scalar values form a 3D surface. Sometimes, especially in conjunction with the description of filters, we are interested in what we call the support representation of a 2D signal. In this representation, we take a birds eye view of the signal and we only represent the nonzero values of the signal as dots in a two dimensional plane. Since the height of the pixels, namely the scalar value, cannot be inferred just by the dot representation.

We often write the value on the signal of the particular location next to the dot. So this plot for instance, represents the two dimensional delta signal. Which is a signal which is 0 everywhere except in the origin where it is 1. And so you have just one red dot here at the origin with value 1. Of course, the most common representation for a 2D signal which is also an image is an image representation. In this case, we exploit the dynamic range of the medium. In this case we have computer monitor, and we know that each pixel can be driven to represent a different shade of gray. And since the pixel values are packed very closely together in space, here for instance we have 512 by 512 pixel values. The density will be high and the eye will create the illusion of a continuous image. So one question that could come up naturally at this point is, why do we go through the trouble of defining a whole new two dimensional signal processing in paradigm?

Can we just convert images into 1D signals and use the standard things that we've used so far?

And of course the sometimes, that's exactly what we do if we think of the printer that prints one line at a time as the paper rolls out, or a fax machine. That's exactly what happens. However, if we do that, we miss out on the spatial correlation between pixels, and therefore the properties of an image will be more difficult to understand.

Let's look at an example.

Here we have a 41 by 41 pixel image and the content of this image is simply a straight line. We will see that the angle of the straight line will change later. What you have in the bottom panel is what we call a raster scan representation of the image. In other words, we go through the lines of image one by one. And with plot the corresponding pixels on this axis. Now for a horizontal line coincides with the n1 axis the resulting

unrolled representation is just a series of 0 pixels. Except when we scan this line at which point we will have

41 pixels equal to 1 and then we will go back to 0. So this is rather simple to understand, but if we change the angle of the line, we see that the representation in an unrolled fashion changes in ways that are not very intuitive. When the angle is small, we have clusters of pixels interspersed with 0s. As the angle increases, the spacing of the clusters changes and also number of pixels per cluster. It's very hard to understand the visual characteristic of the line from the position of the clusters and the number of pixels.

After we passed the 45 degree angle we will have collections of single pixel clusters. And the spacing of these clusters will change in even more subtle ways according to the angle of the line. Finally when each line that is coinciding with the n2 axis, we will have single pixels that are separated by 40, 0s.

Because as we scan the image, we will hit in on 0 pixel, and then we will have to go 40 pixel before we head to another one.

This simple example should convince you that a full 2D representation is necessary to best describe and interpret an image signa. Just like we did for the 1DKs, here are some basic signals.

The first one we have already seen in passing is the delta, the impulse. Which is 0 everywhere except in the origin, where it is equal to 1. And the support representation is like so. The two dimensional rect signal is defined by two parameters, which we may call the width and the height of the rect. And it is 0 everywhere except in a rectangular region which is defined by those values of the n1 index. They are smaller than capital N1 in magnitude and those values of the n2 index, they are smaller in magnitude to the capital N2. And of course it looks like this.One fundamental property for two dimensional signals that has no equivalenct in one dimension, is that of separability. Now, separability simply means that we can write a two dimensional signal as the product of two independent 1D signals, defined on indices n1 and n2. So the delta signal for instance, is fully separable because it is just a product of 2 delta functions applied to both indices.

And similarly, the rectangular function is again, the product of two one dimensional rect functions defined over N1 and N2.Separability is a fragile property in the sense that we can have simple transformations or
linear combinations of separable objects which are no longer separable. In this picture here you have a square, for instance, which is simple a rect function with equal width and height. But which is rotated by 45 degrees. This signal is now separable. It is expressed as 1 when the sum of the indices in magnitude is smaller than capital N and 0 otherwise. But there is no way that we can express this signal as the product
of two elementary 1D signals. Similarly, the difference of two rect signals which are separable to begin with,
gives rise to a non-separable frame-like pattern as in the picture. Separability does not represent a natural property of images. But it is very important from the computational point of view. We can understand that fully if we consider the two dimensional convolution, which is really a simple straightforward extension of the one dimensional case. The convolution of two sequences in discrete space is simply the sum for
the first index that goes to minus infinity to plus infinity. Of the sum for the second index that goes from minus infinity to plus infinity.
Of the first sequence times the second sequence. Space reversed and centered in n1 and n2. So we just have a doubling of indices with respect to the 1Dks. If one of the signals is separable, then we could show that the convolution can be split into two one dimensional convolutions as shown here. So if we assume, for instance, that h is separable in the convolution of h of n1 n2 with x of n1 and n2, then the convolution can be performed in two steps. First we could involve x with h2 of n2 using only n2 as the free variable,
and then we can involve the result with h1 of n1. The consequences of this split are extremely important when we consider the computational requirements of the

convolution operator. If h[n1, n2] is a finite-support signal, and the support is of size capital M1 times capital M2. And this by the way is going to be the standard setup for FIR filtering operations in 2D. A non-separable convolution will require M1 x M2 operations per output sample. Whereas a separable convolution will require M1 + M2 operations per output sample, which is generally much, much smaller than the number of operation required in the previous case.We will talk about image manipulations and in particular we will consider a class of transformations called affine transforms.

And we will see how to implement affine transforms for the class of digital images using the bilinear interpolation method. An affine transform is a mapping that reshapes a coordinate system. In our case we're talking about images, so we're interested in the mapping from R2 to R2. An affine transform is defined in terms of a two by two matrix and a translation vector. And the new coordinate pair is given by the multiplication of the matrix times the original coordinate pair minus the translation vector. In compact form, we will express it like so. Let's look at some examples of affine transform, a very simple one is translation. The matrix here is the identity matrix and the translation vector specifies the new origin of the plane. So if we have this simple image here and we apply a translation, we would get this simple result.

Scaling is an affine transform where the matrix is a diagonal matrix, and the diagonal entries are the amount of stretching or contraction of each axis independently.

In this example, for instance, we have a scaling where clearly a1 is equal to a2, and both are less than 1, because the resulting image is larger. Rotation is another affine transform, we have seen this before. The structure of the matrix is like so. We have cosine of theta on the diagonal and sine of theta and minus sine of theta on the anti-diagonal. A pure rotation leaves the origin in place and, as an example,

here is a rotation by an angle theta that is clearly larger than pi over two. And here, for instance, you have a rotation by pi.

Flips can be both horizontal or vertical and they simply swap the direction of an axis. In both cases, the matrix is diagonal. And in the case of a horizontal flip, the first element of the diagonal would be -1 and the second one is 1. Vice versa for a vertical flip, the second access gets flipped, and so the second element of the diagonal would be -1. Here in the image we see horizontal flip. Shear is a stretching of the plane in either the horizontal or vertical direction. For this case of affine transform, the matrix is either upper triangular or lower triangular. You have 1s on the main diagonal, and according to whether the shear is horizontal or vertical, you have a parameter s on the upper corner or on the lower corner. The effect of shearing is like so. In this case you have a horizontal shearing, when the horizontal axis is stretched.

If we now try to apply an affine transform to a digital image, we run into a fundamental problem. Remember, a digital image lives in discreet space.

In other words, pixel coordinates are pairs of integers, they live in Z2.

The affine transform on the other hand, maps this original coordinates onto a point in R2, which means that the new pair of coordinates can lie anywhere on the plane and, in particular, it will most likely lie in between individual pixel coordinates. To understand the situation better, if this is the regular grid of pixels in Z2, and this is the result of the affine transformation, we would like that each point here on this grid be mapped to a point on this grid here. But unfortunately the affine transform will probably map this point

here in between.

The first step is to consider the inverse transform. So instead of going from the original image to the destination image,here we draw it again, we start from the destination image, for which we know we want points to lie on the grid. And we compute the originating point from the original images. So if this is the original image, this is a transformed image, we take each point on the grid and we find the point it would come from with the inverse transform. Now just like before, this point will not lie on the original grid.

But here's the second step. This pair of coordinates that we found with the inverse transform, we will express as two integer coordinate, eta1 and eta2, plus two correction factors, tau1 and tau2, where tau1 and tau2 are strictly less than 1. So it is a sub-pixel correction factor. So again if this is a magnified view of the original grid, our transform gives us a point here. We will find that this point t1, t2 is equal to a point on the grid eta1, eta2, plus a correctional factor tau1 here, and a correction factor tau2. So the question now is how to find this sub-pixel value here in the middle of the grid. An the idea is to interpolate from the four neighboring pixels. When the interpolation scheme between neighboring pixel is the linear interpolator, this strategy goes under the name of bilinear interpolation.

So let's see how this works. We have our reference point x(eta1, eta2) and we also have the three neighboring points, like so Remember, our sub-pixel value, as computed before, falls here in the middle, and it is expressed as a pixel at horizontal distance of tau1, and a vertical distance of tau2 from the reference coordinate eta1 and eta2. So to compute its value, we first interpolate in the horizontal direction, and we find two sub-pixels values at vertical coordinate eta2 and eta2 + 1, both at a distance of tau1 from eta1. And then once we have these two values, we compute a vertical interpolation at a distance of tau2, like so, and this gives us our desired value.

Frequency analysis

We only consider finite-size images and therefore the only frequency analysis tool that we need is a two-dimensional extension of the DFT. In this lecture we will see how the concept of basis vector can be extended in a straightforward way to two-dimensional signals and how the DFT formula extends immediately to the space of (N\times M)($N \times M$)-sized images.

One very notable fact in image processing, however, is that Fourier analysis does not play such a major role. Most of the information in an image is encoded by its edges (as

we learn in infancy from coloring books), but edges represent signal discontinuities that, in the frequency domain, affect mostly the global phase response and are therefore hard to see. In fact, the much larger importance of the phase over the magnitude in the DFT of an image is a remarkable difference with one-dimensional Fourier analysis.

We will first define the DFT for two dimensional signals.

And then we will look at the amount of information that we can extract from the magnitude and the phase of the DFT of an image. Fourier analysis of two dimensional signals can be developed exactly as we did for the one dimensional case. Since here we're concerned mostly with digital images, namely finite support two dimensional images.

We will only review the definition of the two dimensional DFT. So let's consider a two dimensional finite support signal of support big N1 times big N2. The DFT is defined as the double sum for the first index that goes from 0 to big N1 minus 1. And the second index that goes from 0 to big N2 minus 1, of the values of the dimensional signal over the support. Times the product of two complex exponentials. Whose frequencies are 2 pi over N1 and 1k1, and 2 pi over N2 and 2K2. The products of these two complex exponentials represents a basis function for the space of images of size N1 times N2.

The DFT can be easily inverted just like we did in the one dimensional case. We take basically complex exponentials with the sign reversed and we repeat the sum. Taking now the DFT coefficients in to the sum. Normalization is by convention applied to the inverse formula. And in this case we have to divide the sum by the product of N1 times N2. It is certainly instructive to look in more detail at the basis functions for the space of N1 times N2 images. These have the form that we have shown before in the DFT sum.

And we can easily prove like in the one dimensional case, that they are orthogonal. There are N1 times N2 basis functions for an image of that size. And so it would be very hard to look at each one of them even for images of moderate size.

But we can try and plot some key representative basic functions to give you an idea.

Of what the building blocks are for an image in Fourier space. We will show these basis functions by plotting the real part only as a grayscale image. Where the value of zero is indicated by black and the value of one is indicated by white.

So here we have the space of 256 by 256 pixel images, and this is one of the simplest basis functions that we can plot. We keep the vertical frequency at zero, and the horizontal frequency spans one period, between zero and 255. So if we were to look at this in a three dimensional space, we could plot. This is the image plane and these are the values of the basis function. And this is a wave, a solid wave, that goes like this.
So, it goes down and then it goes up again. And here we have the white part and here we have the black part. We can invert the roles of the vertical and horizontal frequency.
And we get an image which is simply a 90 degree rotation of the previous one. We can increase the horizontal frequency at this point for k1 equal to 2. We will have that the basis function spans two periods.Over the support of the image And if we swap the roles of the frequency, we obtain, again, We can increase the frequency even more, and
the density of these bands will increase correspondingly. Whenever the vertical frequency and the horizontal frequency are the same, the bands will be angled at 45 degrees. And by varying the frequencies, we can obtain a wide range of different angles for the bands.
The good news is that the two dimensional DFT basis functions are separable. And so the DFT can be computed in a separable way. In particular, we first compute a 1D-DFT along the columns. So if this is our image of size N1 times N2 we first compute N1 DFTs of size N2 along the columns. And once we're done with that, we compute 1D-DFTs along the rows. So computationally speaking we first need to compute N1 DFTs of size N2. We know that we can use the FFT algorithms. So the cost will be N2 log in base 2 of N2. And then we need to compute N2 FFTs of size N1. So that would be N1 log in

base 2 of N1. Which is much less than N1 N2 squared, the cost of implementing a two-dimensional DFT directly from the equation.

We can also express the 2D DFT in matrix form. For that we need to express first of all the signal, the two dimensional signal as a matrix. This is very straightforward because an N1 times N2 image is simply an N1 times N2 matrix. There is only the technicality that the orientation of the rows is inverted with respect to the Cartesian notations. So for instance, in the Cartesian plane N2 would go from, say, zero upwards. And this is our image. But if we express this in matrix notation, this element of the matrix normally is 0, 0. So there's a flipping of the vertical axis but this is just a technicality. You will also recall the N x N DFT matrix that we saw in Module 4.2. This is a standard DFT matrix of size N where W, recall, is simply e to the minus j 2 pi over capital N. With this notation in place let's look at the DFT formula once again. The inner summation is simply the product of the DFT matrix of size big N2 times the signal matrix. We call this intermediary matrix capital V and capital V belongs to the space of N2 times N1 matrices. Then the outer sum can be expressed as the right product of the matrix V we just defined times the DFT matrix of size capital N1. And so the resultant signal is a matrix that collects the DFT values for the image. In compact form, we can express the two-dimensional DFT as the product of a DFT matrix of size N2. Times the image times a DFT matrix of size N1.

So now we know how to compute a two-dimensional DFT. Well, can we look at one?

We could try and plot the magnitude of the DFT since the DFT is a two dimensional signal. And therefore, it can be interpreted as an image. But, this wouldn't work. Because the dynamic range of the DFT is way too big for either a monitor screen or a piece of paper to represent. We wouldn't have enough grayscale level to show the details.

So we could try and normalize the values by dividing the DFT magnitude by its maximum value. And yet that wouldn't work either. Because for images on average the

distribution of the magnitude of Fourier coefficients follows a curve like this one. So what we have here is a few outliers here that really go above the average values of the coefficients. And some tail outliers here that drive the value down. What we're interested in is this band between the extrema.

So we take a two-step approach, first we remove the flagrant outliers. For instance the value of the DFT in 0, 0 is simply the sum of all pixel values. Now for grayscale images where all pixels are positive values between zero and one, say. This will be definitely a large value with respect to any other coefficient. And then to remove the tail, we use nonlinear mapping. For instance, we use a curve like x to the power of 1/3 after normalizing all values between zero and one. So for instance, if this is the range between zero and one the nonlinearity will map this range like so. Which means that the tail outliers that we've seen before will be squished in a band that is very close to 0. And so they will appear as black pixels. And the rest of the values, those that we're interested in, will occupy the bulk of the dynamic range of the median.If we do that, we obtain something like this. So our little dog has been transformed in this cryptic image here. Where if we look very closely, we can identify for instance some lines here and here. That correspond to clear lines in the image captured by some of the basis functions that have these orientations. But by and large it's very, very difficult to understand what's going on.We get a confirmation of our suspicion if we try to invert the Fourier transform starting from the magnitude only. So we take an image, we take the Fourier transform, we discard the phase information. And then we do an inverse Fourier transform from the magnitude. What we get if we start with the dog picture is this picture here. Where we are absolutely unable to guess the original picture. What is more interesting is that if we do the same thing but keep in just the phase, in other words we throw away the magnitude. We set the magnitude uniformly at one but we keep the phase information

and then we do an inverse Fourier transform. The idea is that most of the semantic information in an image is contained into its edges.

Edges are points of discontinuity in the gray level signal. In 1D you would represent an edge as a transition from, say a white level to a dark level. This transition is what the eye perceives as an edge.

Now, edges therefore are very localized in space. And they're not captured by the DFTs magnitude, which represents the distribution in frequency of the energy of the whole image. In order to obtain an edge from a combination of Fourier basis functions on the other hand. Their phase have to be aligned in a very precise manner. And that's why the phase information captures the intelligibility part of an image. A consequence of this is that the global DFT of an image is only of limited use in image processing.

Image filtering

Introduction

In this lecture we will explore the concept of filtering an image. To keep matters simple, we will limit ourselves to FIR filters.

There are two main types of linear, space-invariant filters that we can define:

1. lowpass filters will blur an image and are used mostly before downsampling (that is, reducing the size) of a picture
2. highpass filters will enhance the edges of an image and are used mostly as the first step in an edge detection algorithm.

Another simplification in image filtering is that, normally, an image is a finite-support signal that's always entirely available. In this sense, the notion of causality loses its meaning and we can design simple zero-phase FIRs.