A Comparison of Texture Segmentation Approaches

CPSC 5990: Final Report

Adam Lefaivre (001145679)

August 9th, 2017

University of Lethbridge

Department of Mathematics and Computer Science

Dr. Howard Cheng

Table of Contents

Introduction and Background

Textures are everywhere around us, and they are important for discriminating various sections of an image from one another. There is no consensus on the definition of texture, however, it is extremely useful for describing an image, especially in terms of how smooth or coarse certain sections of an image are [1]. A texture must have some tactile attributes, and will likely have some repeating pattern in terms of intensity variations within an image. We can call this repeating pattern a sub-pattern of an image which is responsible for demonstrating some repetitive aspect of an image [2].

Texture analysis is the use of various mathematical approaches in order to characterize spatial variations within imagery as a means of extracting information. Texture analysis can be performed three different ways: statistically, structurally, and spectrally. Statistical techniques explaining a texture in some quantitative way, such as how grainy a texture is, for example, structural techniques describe textures using image primitives, and finally spectral techniques are based on Fourier spectrum properties, such as detecting periodicity in an image [1]. A significant technique for texture analysis is the construction of the features of an image [3], these are often represented in the form of feature vectors, which are useful for other applications, such as texture segmentation, which we will now define.

Texture segmentation is a problem in image processing where regions having consistent texture are identified [4]. A constant texture is given by a region in which certain statistics of a region in an image are somewhat constant or periodic [5]. Identifying these constant regions is done by using some texture analysis on an image in order to construct a set of features that is then classified. A classification method is then performed in a supervised or unsupervised manner, depending on whether or not there is some prior knowledge of the various texture segments.

The judgment of how well an image segmentation approach performs is not well-decided on, and often falls into two categories, one is where some objective statistical measures of how well an image is segmented is used, and the other is by emulating, empirically, the way the human visual system perceives information [6]. For our purposes, we will simply observe how well one of the texture segmentation approaches segmented a given input image.

The applications of texture segmentation are far reaching, and are used in areas such as remote sensing and GIS, and machine vision. In remote sensing and GIS, aerial photos of certain areas can be identified in an automated fashion, in order to detect the characteristics of certain landscapes [7]. Another area where texture segmentation is found to be important is in machine vision, since it is true that texture segmentation is certainly useful for emulating natural human vision [6]. This is certainly the case for the papers that will be discussed, as the approaches were inspired by the imitation of the human visual system.

Two textural segmentation approaches will now be discussed. They differ mainly in the method of constructing the initial feature images, however, the implementation afterwards is very similar between the papers. Computing the features that are to be considered by some classification algorithm is then of interest.

Description of Algorithms

Unsupervised Texture Segmentation Using Gabor Filters

We will first examine the paper by Jain and Farrokhnia, *Unsupervised Texture Segmentation Using Gabor Filters* [4]. As mentioned, this approach is inspired by a theory of using multiple channels for filtering, in order to represent how the human visual system works. The main idea that the theory proposes is that the human visual system breaks an image into several filtered images, each of which contains some intensity variations for a certain narrow band of frequencies and orientations. The mechanisms used by the mammalian brain to interpret an image can be seen as various band-pass filters, where particular frequencies are kept in some filtered images, and other frequencies are rejected. This paper implements a similar approach to the theory proposed by Campbell and Robson, such that the original image is broken into feature images, also of various frequencies and orientations.

The multi-channel filtering approach used by Jain and Farrokhnia is said to be multi-resolutional, meaning that properties in the image that are of interest, especially in texture segmentation, happen to simultaneously appear on different scales in an image. This is especially true of the filtering method that is proposed by Jain and Farrokhnia, as we will discuss below. One other point to note is that the processing can be done with just gray values in the image, which makes the approach slightly more simplistic [8].

The issues involved with the multi-channel filtering approach, are noted by Jain and Farrokhnia as the characterization of the channels, and how many of them we need, how to extract the features, how dependent channels are on each other, and the integration of the texture features such that some segmentation is produced. These issues are addressed by their texture segmentation algorithm. Which includes the steps:

1) Subjecting an input image to a series of Gabor filters to produce a set of filtered images.
2) Applying a nonlinear transducer to the set of filtered images to get a series of response images, whereby the nonlinear transducer behaves as a blob detector. Note that a blob is just some region where some property is constant, or in common with nearby pixels [9]. The size of the window used to apply the transduction technique is determined with respect to the radial frequency that the Gabor filter in the previous step is tuned to.
3) Applying a local energy computation to the smooth the response of the activation function.

4) Combining feature images and applying a clustering scheme on the feature images to present a segmented image.

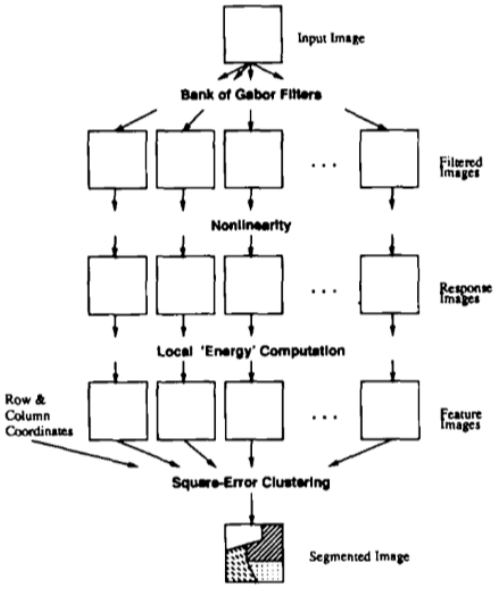

Diagram 1 - The algorithm used in both papers and in other texture segmentation tasks

Finally, the performance of the algorithm is determined by evaluating the segmentation of textures on various images, including images with indistinguishable second and third order statistics [10]. The multi-channel filtering techniques that are proposed by this paper, had been used before in various related problems, such as in texture classification. Similarly, Gabor filters had been used in previous research, but Jain and Farrokhnia's approach was seminal for the further development of texture segmentation using Gabor filters.

The Gabor filters used by Jain and Farrokhnia are real-valued and even-symmetric. Recall that an even function is such that:

$$f(-x) = f(x)$$

Which is symmetric to the y-axis, and also corresponds to the cosine component of a Fourier expression, so we see that Jain and Farrokhnia's definition of a Gabor filter is given as:

$$h(x,y) = \exp\left\{-\frac{1}{2}\left[\frac{x^2}{\sigma_x^2} + \frac{y^2}{\sigma_y^2}\right]\right\}\cos(2\pi u_0 x)$$

Where it's Fourier domain counterpart is also given as:

$$H(u,v) = A\left(exp\left\{-\frac{1}{2}\left[\frac{(u-u_0)^2}{\sigma_u^2} + \frac{v^2}{\sigma_v^2}\right]\right\} + exp\left\{-\frac{1}{2}\left[\frac{(u-u_0)^2}{\sigma_u^2} + \frac{v^2}{\sigma_v^2}\right]\right\}\right)$$

σ is then defined as the spread of the x and y components, as in the standard deviation of a Gaussian distribution.

Where

$$\sigma_u = \frac{1}{2\pi\sigma_x}, \sigma_v = \frac{1}{2\pi\sigma_y}, A = 2\pi\sigma_x\sigma_y$$

The spatial domain representation of the Gabor filter then has two components to discuss, the exponential component is similar to a two-dimensional Gaussian function, and so the greater the spread of the function the less the filters effects are localized to the central point. The periodic component contains the value u_0 which is the central frequency, if it is a large value then the cosine component has a higher frequency. Gabor filters could be seen as wavelet transforms, or rather, band-pass filters with variable size windows and at specific radial frequency and angular orientation combinations. This is because they are similar to a Gaussian curve, but also have some spatial center at some *(x, y)* coordinate. The wavelet transform based on the Gabor wavelet is redundant, and orthogonal decomposition is not a resultant property. The Fourier domain representation identifies the particular frequencies in the image that are then excited, affected, or modulated by the Gabor filter, and as such, these frequencies can be referred to as Modulation Transfer Functions (MTF). So, when setting the MTF of a Gabor filter to a certain value, we are specifying the magnitude by which the Gabor filter affects that particular frequency component [11].

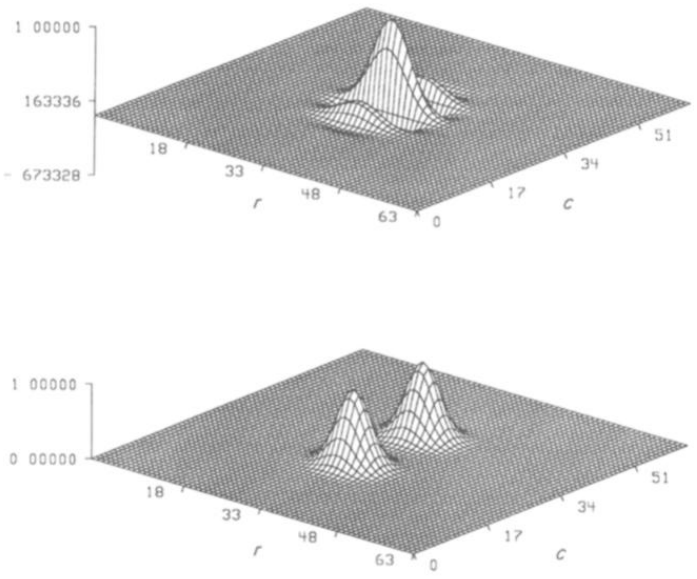

Diagram 2 - A Gabor filter with 0^o orientation and 8 cycles per image width, centered at (r, c) = (32, 32), and its MTF.

What we notice with the Gabor filter expression and its Fourier transform pair is that σ both affects the spread of the filter in the spatial and frequency domains in similar ways. However, as seen above, the σ value in the spatial domain is inversely related to the σ value in the frequency domain. Gabor filters are also said to have optimal joint resolution, meaning that features are extracted in both the Fourier and spatial domain with the optimal simultaneous resolution [12]. This comes from the fact that the filter behaves in a similar manner in both domains, whereas most filters this is not the case, and as such, the bandwidth

in the Fourier domain, and the width of a filter in the spatial domain are inversely related. This provides some detail as to why Gabor filters are a useful filter for the problem of texture segmentation.

The frequency and orientation bandwidths are respectively given by:

$$B_r = log_2\left(\frac{u_0 + (2\ln 2)^{\frac{1}{2}}\sigma_u}{u_0 - (2\ln 2)^{\frac{1}{2}}\sigma_u}\right) \text{ and, } B_\theta = 2\tan^{-1}\left(\frac{(2\ln 2)^{\frac{1}{2}}\sigma_v}{u_0}\right)$$

Notice once again, that we see *σ* affecting the bandwidth, such that a higher value of *σ* affects the orientation bandwidth especially. Note that the frequency bandwidth can be negative, for small *σ* values, but the orientation bandwidth is always defined as positive. A negative frequency bandwidth has to do with the response that that filter has in the Fourier domain, such that, it is summed with other sinusoids in the frequency domain to describe an images frequency representation. B_r is measured in octaves and B_Θ is measured in degrees. The measurement of octaves is used in other areas of signal processing, especially in audio, whereby an octave range is given by a 2:1 frequency ratio, and the bandwidth by $log_2(f2/f1)$.

In Jain and Farrokhnia's implementation, they used just four Θ_0 values, in order to reduce some computational burden, being: 0, 45, 90, and 135 degrees. They also mention that even finer orientation values are recommended, as this, by necessity, would improve the unsupervised classification, however, this results in more feature images which leads to a greater complexity. Recall from the discussion above, that the Θ value is responsible for the directionality of the Gabor filter. Modifying the Gabor filter's Θ value will be imposed upon the whole image, this is due to the periodicity property in the Frequency domain.

Jain and Farrokhnia also give a range of octaves for the radial frequencies used for their implementation of Gabor filters. The range of octaves is given by:

$$1\sqrt{2}\,, 2\sqrt{2}\,, 4\sqrt{2}\,, \ldots, and\ (N_c/4\,)\sqrt{2}\ \ cycles/image\ width$$

Note how each of these frequencies is double the last frequency, this is the property of octaves that was previously discussed. The maximum value is chosen because this frequency will still be within the image width, and accounts for Fourier spectrum periodicity. The total number of filters in the set is given by $4log_2(N_c/2)$, Jane and Farroknhia also note that the $1\sqrt{2}$ and $2\sqrt{2}$ radial frequencies could be left out of the filtered image acquisition step, since the resulting wavelength is larger, and therefore account for textural information that likely does not vary throughout the image. The use of filters with the given radial frequencies were enough to have plenty of coverage in the frequency domain, therefore enough frequencies could be excited, to use the filtered images as feature images to cluster in later steps in the algorithm.

Jain and Farroknia set the mean of each filtered image to zero, thus resulting in an MTF at *(u,v) = (0,0)* to 0. This is because the DC component of an image, or the frequency at

the *(0, 0)* coordinate corresponds to the mean, and if the filtered image has zero mean, this, by definition, implies that the Gabor filter had no effect at the (0,0) coordinate. This has to do with a processing step related to removing DC bias, which causes the frequency domain representation to be centered at zero, and therefore centers the data which assists with the nonlinear transducer step that shortly follows.

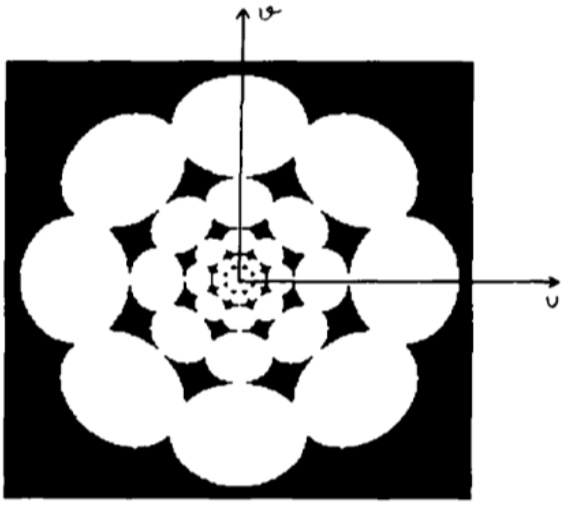

Diagram3 – A set of Gabor filters centered at (u, v) = (0,0)

Now, that the properties of the Gabor filter, and the acquisition of the filter images has been discussed, we can proceed with a discussion on which filtered images are maintained throughout the rest of the algorithm. This is used to limit the running time of the algorithm, especially before applying a transducer step, and clustering algorithm. Jain and Farrokhnia present two filter selection methodologies, one is based on the error in the partial reconstruction of the image with various filters, and the other is based on energy calculation. We will now consider the first method of selecting filtered images. $s(x, y)$ is said to be the image obtained by adding all of the filtered images together into a single image. $\hat{s}(x, y)$ is then the array obtained by using a single filter on the input image. The error of using $\hat{s}$ instead of just using s is given as:

$$SSE = \sum_{x,y} |\hat{s}(x, y) - s(x, y)|^2$$

In both filter selection mechanisms, Jain and Farrokhnia define a coefficient of determination (COD) denoted as R^2 which is used as a threshold, as we will see below. This R^2 calculation is used to determine just how many filtered images are needed to account for some percentage of the intensity variation. For the purposes of their approach, a value of 95 percent was chosen. This is given by the expression:

$$R^2 = 1 - \frac{SSE}{SSTOT}$$

Where:

$$SSTOT = \sum_{x,y} |s(x, y)|^2$$

Note that before this filter selection takes place, recall that the mean of each image output is set to zero, giving an MTF of zero at the DC component. The process Jain and Farrokhnia use to choose their filtered images is given by the following steps:

1. Find the filtered image resulting in the smallest SSE value.
2. Try to add another image with the smallest one found, if it approximates $s(x, y)$the best, then keep it
3. Try picking up another filtered image by repeating the second step. We can continue this until $R2 \geq 0.95$.

At this point, there are enough filtered images that any further processing will be both more efficient and accurately portray the results of applying all of the filters on the image.

As stated previously, there are two methods of determining which filtered images to keep. The first has been discussed; the second is the preferred method of implementation according to the authors. It is based on Parseval's theorem, and in this case, refers to the fact that the energy of the Fourier Transform representation of an image is the same as the energy of that image in the spatial domain. The overall energy of the original image can be approximated by the summation of the energies of each filtered image E_i giving:

$$E \approx \sum_{i=1}^{n} E_i$$

And by Parseval's theorem, we get:

$$E_i = \sum_{x,y} |r_i(x, y)|^2 = \sum_{u,v} |R_i(u, v)|^2$$

Then, after sorting the filtered images by their energies, for some subset of filtered images S we can simply add filtered images to this subset, where:

$$R^2 \approx \frac{\sum_{j \in S} E_j}{E}$$

We can then choose enough images to have the R^2 value surpass 95 percent. Note that R^2 can be set higher if need be. In the case of more detailed texture segmentations, any reduction of intensity information may yield undesirable segmentation. Note that the filter bank is comprised of various angular orientations as well, so eliminating some of the image's response at various orientations is not desirable when seeking a highly detailed segmentation output.

After filter selection has eliminated some of the images in the filter bank, features can then be created. Recall that this is the last step before feature vectors are created and clustering is executed. This step is quite important, as it is responsible for taking the filtered Gabor images, and creating features that can be used to determine the segmentation of textures in an image. Jain and Farrokhnia present a nonlinear transducer which acts as a blob detector. The reasoning is for psychovisual purposes, whereby local features are more easily identified, these local features are called textons, and are preattentive to the human visual system. These textons are comprised of tiny malleable geometric structures [13]. It is

mentioned that the function used as a nonlinear transducer is similar to a sigmoidal activation function, meaning that the function has a curve of an "S" shape, is real, monotonic, and differentiable [14]. This is certainly the case of the activation function given by Jain and Farrokhnia:

$$\psi(t) = \tanh(\alpha t) = \frac{1 - e^{-2\alpha t}}{1 + e^{-2\alpha t}}$$

Where it is applied as such:

$$e_k(x, y) = \frac{1}{M^2} \sum_{(a,b) \in W_{x,y}} |\psi(r_k(a, b))|$$

Where $e_k(x, y)$ is the final, filtered image, before clustering is applied, also, $W_{x,y}$ is a window of size *MxM* centered about a pixel *(x,y)*, and ψ is the nonlinear sigmoidal activation function. Note that the transducer can be applied to every pixel in the image, and then the averaging window of width and height M can be applied afterwards.

For the filtered input images, features are exaggerated once this transducer is applied, since it is nonlinear. Due to the threshold-like nature of the sigmoidal function, more prominent qualities are exaggerated, and lesser prominent ones become even less important. For this reason, it is known as an activation function. This assists in the clustering of features because more weight will be associated with prominent features, and a smaller weight will be associated with less prominent features. This function was also of importance due to its applications in neural networks and the importance Jain and Farrokhnia placed on emulating psychovisual properties using their classification pipeline, this is due to how it emulates the firing of a neuron.

After applying the hyperbolic tangent function to each pixel, some Gaussian smoothing is done to the image. This results in an overall representation of the activity in the window of the filter. Smaller averaging windows are crucial for more detailed texture segmentation, whereas larger regions containing constant textures can be smoothed in such a way that that the features of a larger texture area are influenced by the more prominent features. The reason for this smoothing window is to spread the activity of the thresholded output (from the tanh function) to surrounding pixels. With a Gaussian window, this activity can be more centralized to the current pixel position in the image, or with a higher standard deviation, can further influence surrounding pixels in the window. σ was found to be:

$$\sigma \approx 0.5T$$

And:

$$T = N_c / u_0$$

Where N_c is the number of columns in an image and u_0 is the center frequency of the Gabor filter. To smooth this function, the approach to handling periodicity in the Fourier domain is to reflect the image at the image boundary. This is a small detail that is of interest, as textural regions at the boundary of an image can be replicated outside of the image boundaries. If the image was allowed to just repeat in the Fourier domain, then some texture on the other side of the image could be smoothed into one another, leading to less accurate results. Note that *σ* was kept to *0.5T* throughout all of the texture segmentation tasks performed.

At this point, all of the feature images necessary for texture segmentation have been acquired. Features are arranged such that the value of each pixel in a single feature image counts as a feature for that pixel coordinate. So, for some pixel at location *(i, j)*, with *N* different feature images we can construct a feature vector, as introduced by [3], and given by:

$$T_{ij} = < F_1(i,j), \dots, F_n(i,j) >$$

In Jain and Farrokhnia's paper, they next describe a clustering algorithm, in which feature vectors are passed in with additional row and column information as attributes. Coordinate information is included in each feature vector, since many pixels near to each other will belong to the same texture segment. Including row and column data in each pixel's feature vector assists in clustering together neighboring pixels. The clustering algorithm that is used is based on the K-means algorithm, except another pass is used to cluster together the clusters that were found in the first pass even further. Jain and Farrokhnia go even further and suggest a method of determining the number of clusters, as well as labeling each pixel, so that the output of the classifier can be compared. This is to check for errors in the labeling given by their segmentation techniques. They then derived some accuracy for each input image for their results. The details of the clustering algorithm, as well as the labeling of each pixel is not of interest, since the focus is primarily on the construction of the feature vectors.

This concludes the discussion of Jane and Farrokhnia's texture segmentation method using Gabor filters. We will now elaborate on the feature extraction methods used by Mihran Tuceryan in his paper *Moment-Based Texture Segmentation* [3]. It is important to recall that most of this algorithm is the same as Jain and Farrokhnia's, the main difference is in the use of moments, as opposed to Gabor filters. There may be some small implementation details in each step that will be compared to the paper just discussed, as some changes are made throughout the steps of the algorithm.

Moment-Based Texture Segmentation

Tuceryan begins the explanation of the algorithm with a means of measuring the success of the texture segmentation algorithm in general. In agreement with Jain and Farrokhnia, Tuceryan states that texture segmentation is only truely successful if the results are in accordance with what the human visual system should perceive as the correct segmentation. In claiming this, Tuceryan further states that the human visual system both performs classification and segmentation, and that the discrimination of textons is rather

important for the problem of texture segmentation. Textons are the basic features of textures, and recall that they were mentioned in the previous paper as being local features of a texture. So, in order for segmentation to occur, these textons need to be exploited in some way, and like the usage of Gabor filters, moments are meant to perform the same type of feature extraction on textons.

The segmentation algorithm used by Tuceryan applies the same steps as in the previous paper. Feature images are derived by finding image moments around every pixel, with some smaller window size, then, the same nonlinear transducer and clustering approach is applied.

The moments of a two variable function are given by:

$$m_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x,y) x^p y^q \, dxdy$$

Where p and q denote the order, such that $p + q = 0, 1, 2, \ldots$ Note that when an infinite number of moments are used, the image is a representation of all of the moments, and the opposite is true. In this way, the calculation of moments is somewhat related to a Taylor expansion, in how a function can be well-approximated with further derivative components. The infinite set of moments are given by $\{m_{p,q}, p + q = 0,1,2, \ldots\}$, and:

$$f(x,y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \exp\big(-j2\pi(ux + vy)\big) \times \left[\sum_{p=0}^{\infty} \sum_{q=0}^{\infty} m_{p,q} \frac{(j2\pi)^{p+q}}{p!\, q!} u^p v^q \right] dudv$$

Lower order moments can be computed, and are often enough to describe the image content. These moments are computed within some region, let this region be denoted by *R*. If the value of this function inside of a boundary is different than outside of the boundary, then there are some geometric properties that are of interest. If *p* and *q* are both set to zero then we basically get the area of the window size back, so we can use this area to calculate the location of the centroid in that window, by finding the quotients, m_{10}/m_{00} and m_{01}/m_{00}, this also corresponds to the "center of mass" calculation of an area. The next two moments correspond to the mean and variance, and with further moments, the elongation and major axis orientation of the region can also be calculated [16]. These are just some of the uses of moments, and are good examples of just how descriptive they are; as such, even higher order moments contain even more descriptive content.

Tuceryan only used lower order moments, such that $p + q \leq 2$ which was indeed sufficient for their texture segmentation purposes, and as mentioned was enough to describe the contents inside each region, and therefore the image. The x_m and y_n components represent normalized pixel coordinates, such that:

$$x_m = \frac{m-i}{\lfloor W/2 \rfloor} \; and \; y_n = \frac{n-j}{\lfloor W/2 \rfloor}$$

And the moment calculation is given by:

$$m_{pq} = \sum_{-W/2}^{W/2} \sum_{-W/2}^{W/2} f(m,n) x_m^p y_n^q$$

Where W is the width of the window, *(i, j)* is the coordinate of the pixel at the center of the image, and (m, n) is some pixel within the window, where:

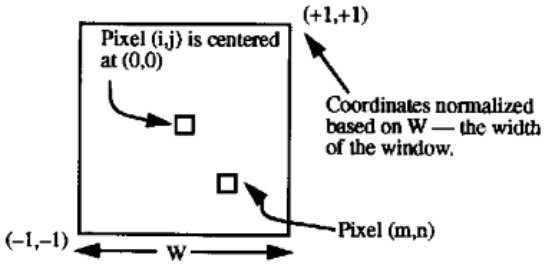

Diagram 4 - How the moment mask is applied to the pixel at (i, j)

This window gets centered at every pixel in the image such that it becomes an image convolution, with each moment filter of size *WxW*. With $p + q \leq 2$ we get the moment masks for convolution as:

$$m_{00} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad m_{10} = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \qquad m_{01} = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

$$m_{02} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix} \qquad m_{11} = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix} \qquad m_{20} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Tuceryan notes that these masks act as detectors of features, much like other convolution masks to detect properties such as edge detectors, like Sobel operator masks, for example. The m_{00} mask returns the total energy in that window, the m_{10} and m_{01} masks respectively detect changes in the *x* and *y* directions by taking the difference of the pixels at the outermost edges of the window, as is typical of edge detection kernels. If there is an edge, these masks will return a larger value for m_{pq}. Similarly m_{11} acts as a diagonal edge detector. The m_{02} and m_{20} masks give the energy of the filter at its edges, and account for the variance in the vertical and horizontal directions. Furthermore, the window size of the image is a tremendously useful parameter. Similar to the size of the Gabor filter, if a larger moment window size is chosen, the ability to detect larger textural changes throughout the image becomes apparent; however, a smaller window size will detect more intricate details.

As similar to the paper on Gabor filters, a nonlinear transducer is used after subtracting the mean from the filtered images. The nonlinear transduction function is the same as explained by Jain and Farrokhnia, except with a different weighting mechanism that was experimentally determined by Tuceryan, the weighting mechanism is $\sigma = 0.01$. In further contrast to Jain and Farrokhnia's approach transduction approach, Tuceryan uses a simple averaging mechanism after applying the hyperbolic tangent calculation to every pixel, as opposed to Jain and Farrokhnia's use of a Gaussian filter. This is done for the same reason discussed in the previous paper.

After the nonlinear transduction is applied, Tuceryan groups the features together in the same way as Jain and Farrokhnia, that is, with each pixel being represented by a vector of features with each moment image's value, at that same pixel location, being one of the features. Pixel coordinates are also included in the feature vectors so that the clustering algorithm benefits from the possible texture similarities of nearby pixels. There is a slight difference in how this paper utilizes the clustering algorithm, such that, the number of clusters is not automatically determined, as it is in Jain and Farrokhnia's paper. Tuceryan also did not check the accuracy of the segmentation algorithm on input images, as Jain and Farrokhnia did. One final and important point about Tuceryan's implementation is that sometimes an extra cluster in addition to the expected K clusters be specified, as to handle border effects presented by convolution and padding.

Implementation Details and Results

Firstly, it must be noted that both papers were implemented in Python version 2.7, with the use of functions from the Open Source Computer Vision Library (OpenCV Library), as well as SciPy, Python Imaging Library (PIL), and SciKit-Learn. Specific functions were chosen from each library because of their ease of implementation and documentation. One final note, both the Gabor filter and moment features were normalized to between 0 and 1 prior to clustering.

Implementation Details: Unsupervised Texture Segmentation Using Gabor Filters

In the implementation of this paper, the OpenCV function *getGaborKernel* was utilized. This function uses the real valued equation:

$$g(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)\cos\left(2\pi\frac{x}{\lambda} + \psi\right)$$

The parameters of this function are listed here for convenience. λ is the wavelength, found by taking the inverse of the radial frequencies given by Jain and Farrokhnia. θ is the angular orientation of the Gabor filter. ψ is added to the sinusoidal component to offset the periodicity and alter the phase. σ is the standard deviation, or spread of the Gabor filter.

Finally, γ is what is known as the spatial aspect ratio, which controls the ellipticity of the Gabor filter [17].

Then, in order to use the radial frequencies as specified in the paper, the wavelength, given by λ, had to be calculated. The wavelength calculations were computed by taking the inverse of each radial frequency. Recall that each radial frequency was given in cycles per image width, so each inverse frequency found also had to be multiplied by the width of the image.

The Θ values were computed by converting the angles given in the paper to radians, and then finally, the rest of the function's parameters were left for the user to choose. To match the exact Gabor kernel used in the paper ψ has to be set to zero, σ_x has to be equal to σ_y and γ has to be set to one. These are the default parameters in the implementation, nevertheless good results were still achieved by trying non-default parameters. We can describe how choosing different parameter values might be beneficial to the enhancement of the segmentation output at this point.

It was found that σ needed to be altered quite frequently in order to provide a less sharp filter that excited more frequencies at the specific orientation and frequency chosen. The γ variable was also left to be altered by the user, in the case that the Gabor filter had to be more elliptic. Ellipticity, in this case, is caused by the spatial aspect ratio, or the ratio of the Gabor filter's height to its width. Increased ellipticity enhances the effect of the filters orientation, so it can be controlled to help segment out more elongated texture regions. ψ was maintained at zero throughout the segmentation tasks, however it, was left as a possible parameter in order to adjust the sinusoidal component of the Gabor filter, in the case that there was a situation where some phase offset provided better segmentation results. Finally, the size of the Gabor filter was left as a parameter without a default value, since the size of the filter is totally dependent on the size and variety of texture regions in the input image. The λ and Θ values were kept the exact same as the paper, in order to match the implementation of Jain and Farrokhnia, otherwise all, of the above mentioned parameters were left to be modified freely.

The kernel given by the *getGaborKernel* function was then smoothed by a factor of 1.5, multipled by the sum of the kernel. This is a common operation following the usage of OpenCV's *getGaborKernel*. It provides some smoothing and helps scale the kernel such that the filtered image bank looks similar to that of Jain and Farrokhni's. If it is not done, the kernel's maximum value will far outweigh its minimum, so it can be seen as a type of normalization as well, this helps even out the response of the kernel while convolution is occurring, since the central pixel in each convolution mask will not be assigned the highest value of the kernel's response. Next, each kernel is convolved with an input image and stored in an array of filtered images. After this, the filter selection scheme used in this paper is called. An additional parameter to have a hard threshold on the maximum number of images is also implemented. This is helpful if a very large image is given as input. This way, if brute force

nonlinear transduction, with a larger kernel size is implemented, then there is the convenience of specifying how many filtered images should be kept.

After the filtered images are selected, the averages of these intermediate output images are removed, thereby following the implementation that the MTF be zero at *(u, v) = (0, 0)*. There is then a normalization step that is not mentioned in the paper (however it was implemented), where each filtered image has all of its pixel responses set between -2 and 2, this is so that the nonlinear transducer defined in the paper has a well-defined response, if the output of convolution is used without this scaling then the hyperbolic tangent function will not act as it should, in that most of the values after convolution (and after the mean has been subtracted) will cause an immediate activation of the transducer, and this is not desirable. Furthermore, the normalization between these values was chosen because it is the "active-range" of the tanh function, that is, when the derivative of the function is certain to be non-zero. After this scaling is done and the transduction function has been applied to every pixel, then this output is subjected to Gaussian smoothing. The window size of the Gaussian function is able to be set arbitrarily large by the user. This controls just how smoothed out the response of the transducer will be. The σ value corresponds to the calculation $\sigma \approx 0.5T$. Finally, feature images with small variances are removed, since a feature image with a small variance at this point will not assist in any clustering tasks, this is also mentioned in Jain and Farrokhnia's paper. This variance threshold was set to a value of 0.0001, as it is in the paper.

Now, the construction of feature vectors and the clustering must be discussed. Once the feature vectors are constructed for each pixel, the row and column values are added, and the features are normalized to have constant variance. The variance was set to a constant by the use of SciPy's *whiten* function. Next, the option to add some spatial weight to the row and column values is offered. This is an extra feature that is added before clustering to amplify the weight of the pixel coordinate. This is just done by a parameter that can be set by the user to increase the normalized pixel data. Finally, SciKit-Learn's K-means clustering algorithm is used for its simplicity and documentation. Note that in the paper, they randomly sample a percentage of pixels to be sent to clustering, but this does not matter here, since there is no problem with the speed of the chosen K-means algorithm and the test images used were all small enough.

The input images that were used as test images were the same pairs of textures from the Brodatz album used by Tuceryan, and Jain and Farrokhnia. The Brodatz album is a book created by Phil Brodatz [18], which is filled with plenty of texture images that have become standard images to test results against for the problem of texture segmentation. Pairs of textures were taken from this album, cropped to two smaller subimages, both of which being *128x128* pixels, and concatenated side by side, creating a new output image of size *256x128* pixels large. So the same four textured image presented by Tuceryan was of size *256x256* pixels. Some results will now be show.

Discussion of Results: Unsupervised Texture Segmentation Using Gabor Filters

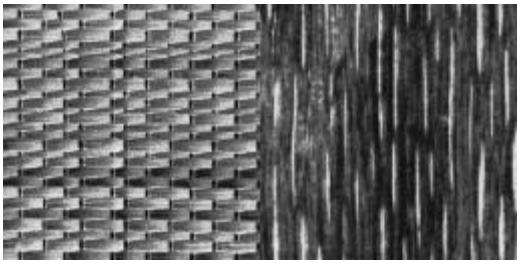

Figure 1a - Input test image with Brodatz ID's of D55 and D68

Figure 1b - The output after clustering, using 14 Gabor filters (not 13), with values $\gamma = 1, \psi = 0, \sigma = 7$, a Gaussian filter size of 31x31, a Gabor filter size of 17x17, and a spatial weight of 1.

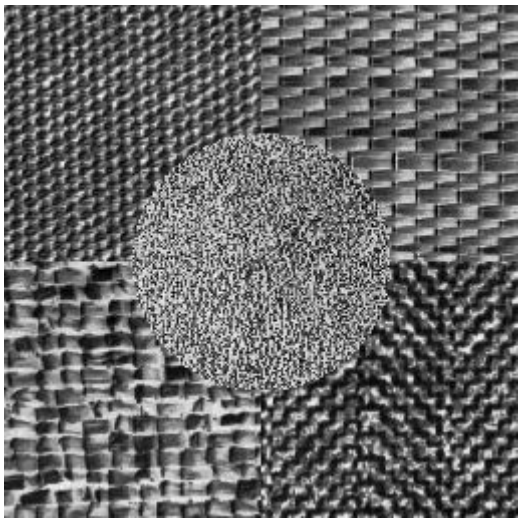

Figure 2a - Input test image with Brodatz ID's of D55 and D68

Figure 2b - Output, of Figure 5 with values of $\gamma = 1, \psi = 0, \sigma = 7$. Using a Gaussian filter size of 31x31, a Gabor filter size of 17x17, and a spatial weight of 1. Using 13 Gabor filters.

Figure 2c - The same as figure 2b, but now with a spatial weighting of 2, instead of 1.

Figure2d -The same output parameters as Figure 2b, except with θ orientation values ranging from 0^o to 150^o in increments of 10^o

Figure2e -The same output parameters as Figure 2b, except with radial frequencies incrementing in a 3/2 ratio (a "just perfect fifth" musical interval), and not in octaves.

Here we observe the output of figures 1 and 2. These examples, and the next one are comprised of all natural images, some of them include wood bark, straw mat, and reptile skin, for instance. Note that $\gamma = 1, \psi = 0$ for examples 1 and 2, which are the default values chosen to match the paper's implementation. Also notice that the Gabor filter size, and the Gaussian filter sizes are quite large. One thing to mention is that these values were chosen by trial and error, because there was no specification of these parameters in the paper, except for the σ value of the Gaussian filter used after the the activation function is applied. The results are quite good, and comparable to Jain and Farrokhnia's. Figures 2d and 2e are of interest, since they result in plenty more Gabor filters and feature images which passed the variance threshold. They demonstrate that although there are more features, this does not necessarily mean the classification will benefit as a result, in fact, it takes much longer to compute, and is a heavier task for the clustering, especially without the use of the R^2 threshold. The results in figures 2d and 2e might be worse than that of 2b because the filters that were chosen after thresholding with R^2 did not provide as much of a spread for the radial frequnecy and angular orientation parameters. If possible, it is prefferable to have an evenly spread set of filters representing the features of the input image.

Recall that for figure 2a as input, with the radial frequency and angular orientation combinations there should be $4log_2(N_c/2)$ filtered images. Since the image width in this case is 256, the number of filtered images should be 28 filters, however, after the R^2 energy threshold has been applied, there are only 17 filtered images that remain, and after the variance thresholding has been applied, there are still 13 feature images. This means that only 13 Gabor filters will be used in the feature vectors. Thus matching Jain and Farrokhnia's filter selection scheme. If there are any differences in the number of Gabor filters used when

compared to Jain and Farrokhnia, like in figure 1b, it is likely a result of the input image. Generally, around the same number of texture features were used for each image as Jain and Farrokhnia mention.

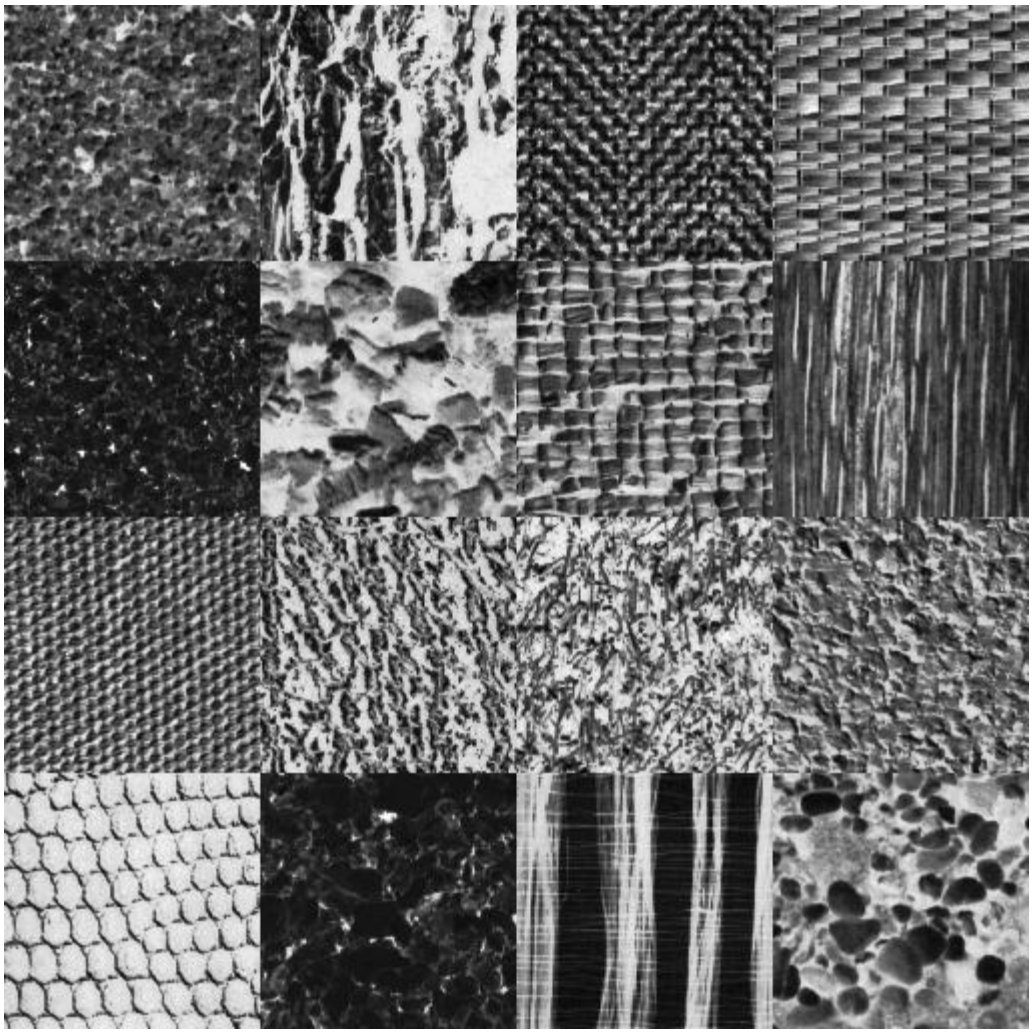

Figure 3a - Input test image with Brodatz ID's of D29, 12, 17, 55, 32, 05, 84, 68, 77, 24, 09, 04, 03, 33, 51, 54

Figure 3b - Output, after clustering of Figure 3a with values of $\gamma = 1, \psi = 0, \sigma = 7$, as well as a Gaussian filter size of 35x35, a Gabor filter size of 17x17, and a spatial weight of 2, using 19 Gabor filters.

This image demonstrates that the segmentation implementation was able to achieve the same results as Jain and Farrokhnia. More spatial weight was needed to make the result even more correct. Even so, it was still successful in determining the different textures within the input image. Note that the two middle textures in the third column are quite similar, so it is reasonable that they were clustered together. In Jain and Farrokhnia's output, some texture regions are clustered together as well.

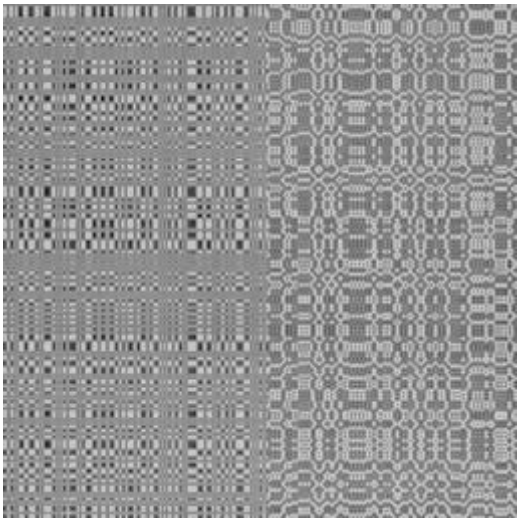

Figure 4a - "Even-Odd" texture pair, a psychovisual texture pair with identical third-order statistics.

Figure 4b – Output of Figure 4a with values of $\gamma = 1, \psi = 0, \sigma = 5$, as well as a Gaussian filter size of 41x41, a Gabor filter size of 7x7, and a spatial weight of 1. Using 19 Gabor filters.

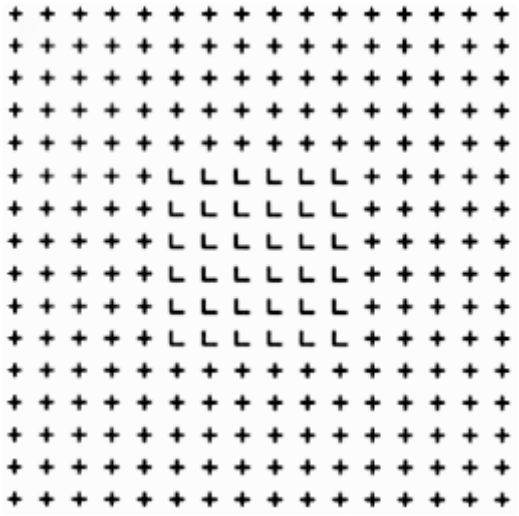

Figure 5a - "L and +" texture pair, a psychovisual texture pair with identical power spectra.

Figure 5b – Output of Figure 5a with values of $\gamma = 1, \psi = 0, \sigma = 7$, as well as a Gaussian filter size of 49x49, a Gabor filter size of 17x17, and a spatial weight of 1. Using 14 (not 19) Gabor filters.

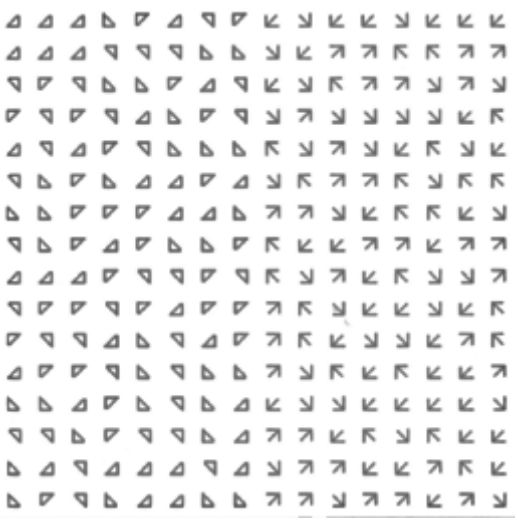

Figure 6a -"Triangle-Arrow", a psychovisual texture pair with identical second order statistics.

Figure 6b- Output, of figure 6a, with values of $\gamma = 0.5, \psi = 0, \sigma = 9$, *as well as a Gaussian filter size of 49x49, a Gabor filter size of 17x17, and a spatial weight of 1. Using 15 (not 17) Gabor filters.*

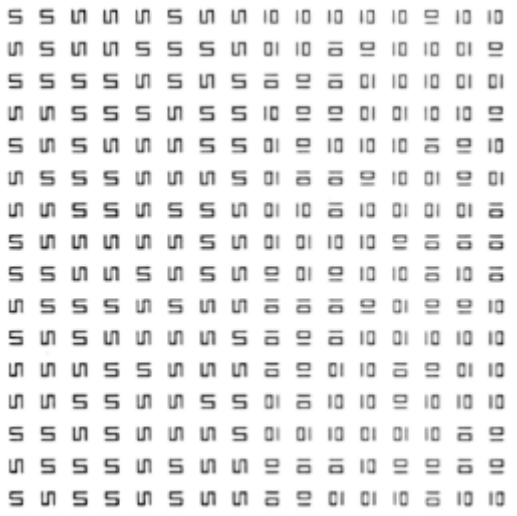

Figure 7a - "S and IO", a psychovisual texture pair with identical second order statistics.

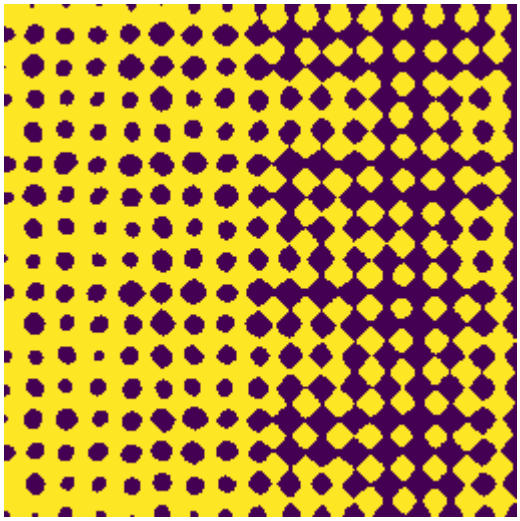

Figure 7b – Output of Figure 7a with values of $\gamma = 1, \psi = 0, \sigma = 7$, *as well as a Gaussian filter size of 7x7, a Gabor filter size of 13x13, and a spatial weight of 1. Using 19 Gabor filters.*

Here we observe that figures 4 to 7 are able to be segmented according to these Gabor filters. The ellipticity is also always set to a circle in accordance with the paper, however various ellipticity values between 0 and 1 were tested, and most times did not provide a substantial difference in the output image. However, it was often the case that the σ value of the Gabor filter be altered, in order to detect more widespread textural properties. The nature of these input images shows us that the Gabor filter is effective in differentiating images where

higher-level statistics become identical. Note that no parameters for each input image was mentioned in Jain and Farrokhnia's approach.

Implementation Details: Moment-Based Texture Segmentation

The implementation first generates all of the *p, q* combinations up to some threshold, this is left as a parameter that the user can change, however the default is set to that specified by the paper, where $p + q \leq 2$. So the user can pick up to any number of moment images. In generating all of the outputs of the moments, the user can also specify the size of the moment filter, and in computing the output of each moment filter, there are two ways that were implemented that produce the same results. The first is by way of brute force, where the moment mask is calculated based on the pixel location and its surrounding pixels, as given in the equations provided by Tuceryan. The second method is an optimized version, where the masks are made prior to the convolution, and not while the convolution is occurring; these masks are passed into SciPy's 2D convolution algorithm. This algorithm was chosen because it is optimized to perform on 2D arrays. This convolution function also has the option of the 'same' mode, which replicates the effects of the brute force algorithm at boundaries and edges, in the case that boundaries are filled with zeros. After both convolution approaches are completed, and before the nonlinear transducer is called, all filtered images are scaled to a value between 0 and 1, just as it is in the implementation dealing with Gabor filters. The averaging mask is then calculated after the hyperbolic tangent function is applied to every pixel, and much like the implementation of the previous paper discussed, the window size of the averaging mask can be set arbitrarily large. OpenCV's *blur* function is used, with the option of "Border Reflect 101", since the averaging window might blur the same texture into the region that it belongs to, if the texture is located near the edge of an image.

Finally, the output images from the nonlinear transducer are sent to be normalized. This paper's implementation differs slightly for this step, by specifying that there also be zero mean for each feature, so this calculation is included in the normalization step, by passing in a flag to do so. The pixel coordinate data is then added, where there is also the option of supplying a weight to the row and column data, just as in the paper implementing Gabor features. The clustering is done using the K-means algorithm discussed in the implementation of the previous paper.

Discussion of Results: Moment-Based Texture Segmentation

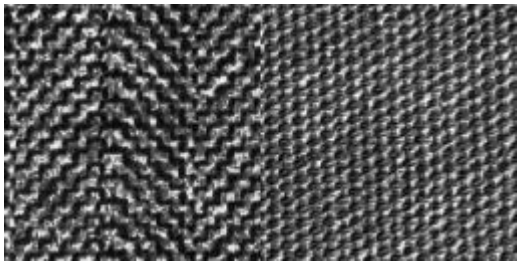

Figure 8a - Input test image with Brodatz ID's of D17 and D77

Note that the exact image for this pair, as given by Tuceryan, was not used. The same texture pair was still used as input. These textures have similar attributes, but in the creation of feature images below, we see how nicely this method performs in distinguishing these textures.

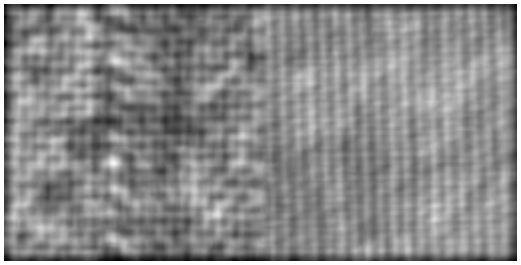

Figure 8b - m_{00} of Figure 1

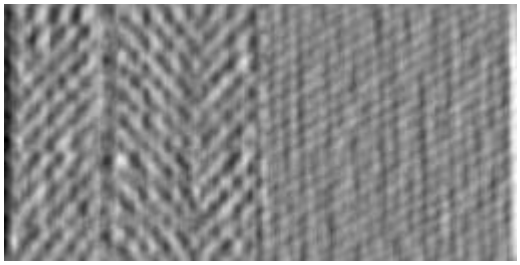

Figure 8c - m_{01} of Figure 1

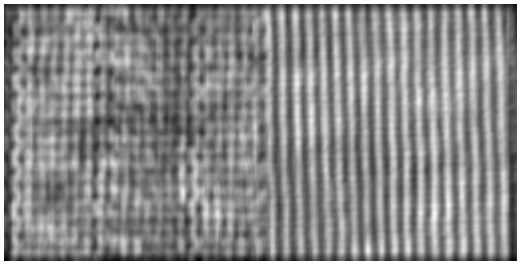

Figure 8d – m_{02} of Figure 1

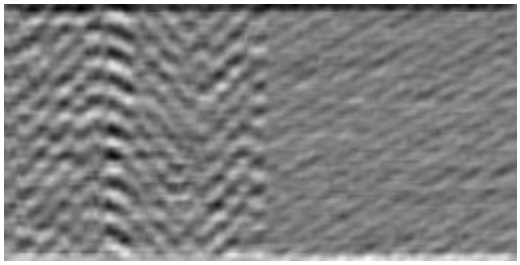

Figure 8e- m_{10} of Figure 1

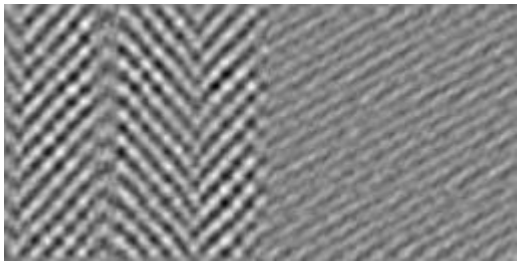

Figure 8f – m_{11} of Figure 1

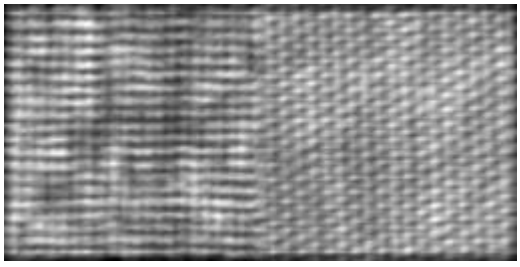

Figure 8g - m_{20} of Figure 1

The intermediate moment images were found with a moment window size of 9x9. These intermediate results are quite similar to the images shown in this paper. There are border effects due to the choice of how boundary conditions are dealt with. The moment kernel window size was not mentioned in this paper, so the intermediate moment images were approximated.

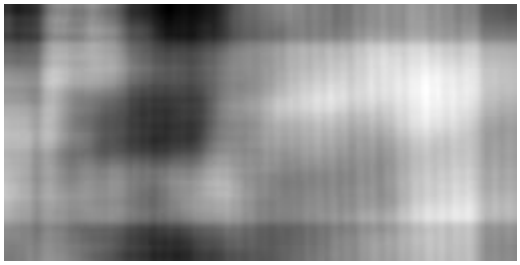

Figure 9a - m_{00} of Figure 8a , after nonlinear transduction and smoothing

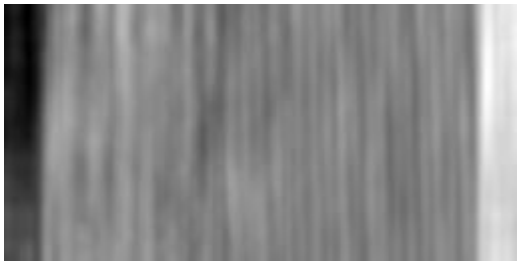

Figure 9b - m_{01} of Figure 8a, after nonlinear transduction and smoothing

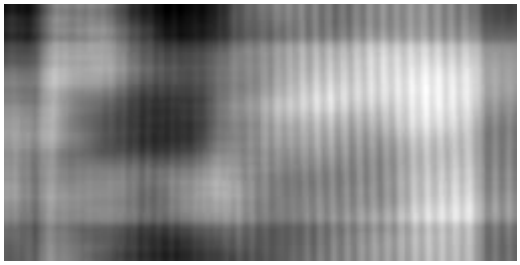

Figure 9c - m_{02} of Figure 8a, after nonlinear transduction and smoothing

Figure 9d - m_{10} of Figure 8a, after nonlinear transduction and smoothing

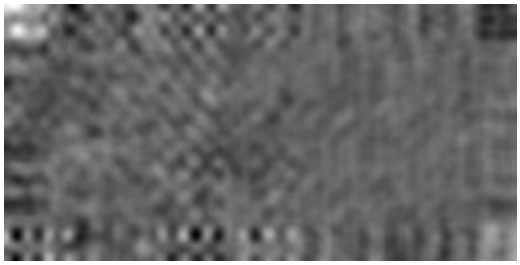

Figure 9e - m_{11} of Figure 8a, after nonlinear transduction and smoothing

Figure 9f - m_{20} of Figure 8a, after nonlinear transduction and smoothing

The feature images before clustering are shown above, they were found with a smoothing window size of *37x37*. Once again, they approximate Tuceryan's feature images for this texture pair. We notice that the features might be clustered easily after this step, due to the differing values in the different texture regions.

Figure 10a – Clustering output with spatial weight = 0

Figure 10b - Clustering output with spatial weight = 1

Figure 10c - Clustering output with spatial weight = 2

Figure 10d - Clustering output with spatial weight = 3

Figures 10a to 10c demonstrate how beneficial the spatial weighting can be. For these examples, an imperfect output was purposely chosen so that the effect of the spatial weight can be observed. For a spatial weight of 1 the features were somewhat successful in differentiating the textures in the image, as one texture is clearly identified apart from the other in Figure 10b. However, the misclassified pixels become classified properly with the proper use of this spatial weight. Often, with too high of a spatial weight the natural segmentation used by the filters is overcome, leading to erroneous results. This is easily done if the row and column data is not normalized before clustering.

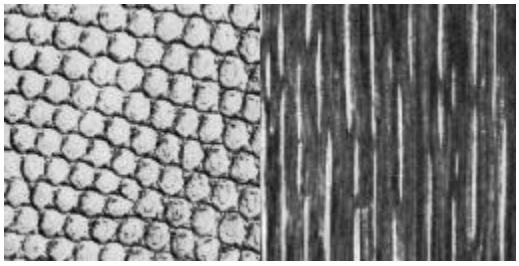

Figure 11a - Input test image with Brodatz ID's of D3 and D68

Figure 11b – The output, after clustering, with moments window size of 9x9, averaging window size of 49x49, and spatial weight of 1.

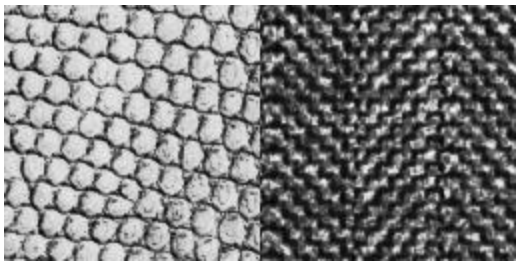

Figure 12a - Input test image with Brodatz ID's of D3 and D17

Figure 12b – The output, after clustering, with moments window size of 9x9, averaging window size of 49x49, and spatial weight of 1.

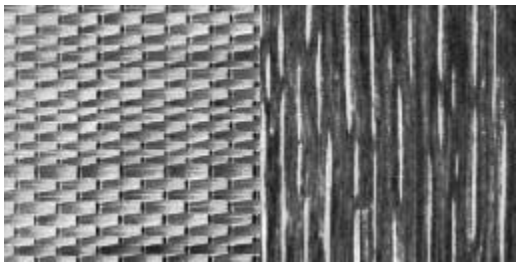

Figure 13a - Input test image with Brodatz ID's of D55 and D68

Figure 13b –The output, after clustering, with moments window size of 9x9, averaging window size of 49x49, and spatial weight of 1.

Figures 11 to 13 are very important, as they demonstrate that the implementation results match those of Tuceryan's, since the exact same parameters were used, and the segmentation output was roughly the same. Furthermore, it demonstrates that this method of texture segmentation is quite useful on simple examples, like discriminating between pairs of textures. Different sized moment windows needed to be used in different contexts. In figures 6 to 7, we notice that a window size of *9x9* was needed for a proper output. This window size and the averaging window size were the parameters that were mostly used to control the segmentation result. The *p* and *q* values were kept between 0 and 2, such that $p + q \leq 2$ since some erroneous output occurred with higher *p* and *q* values, suggesting that higher moments were unnecessary in the segmentation routine. Also, with too many moment images the computation becomes slower. The classification might be affected because higher order moments will consider trends that are not localized to a specific texture region, as show in figure 15a. This is why Tuceryan firmly states this threshold.

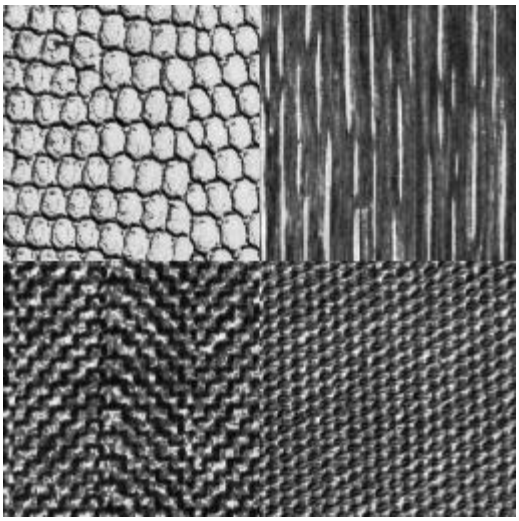

Figure 14a - Input test image with Brodatz ID's of D3, D68, D17, D77 (left to right, top to bottom).

Figure 14b - Its output, after clustering, with moments window size of 9x9, averaging window size of 15x15, and spatial weight of 1.

In the above results, we note that the output is not completely comparable to Tuceryan's, possibly due to the difference in input images, and the difference in parameters used. Note that there are no parameters given in the paper for this example. One other point of interest is the jaggedness of the lines in the output image. This is due to the choice of the averaging window, in that it is a smaller size compared to previous examples. This result is quite impressive, considering there is only a spatial weight of 1.

Figure 15a - The same output as figure 8b, except now with a p + q threshold of 3. The output was similar with higher threshold values.

Comparison Discussion

The results shown in the previous section provide some insight into how these texture features performed. Both of these algorithms performed decently in the segmentation tasks, with some higher amount of accuracy. However, there are most likely newer and more accurate approaches that have been found. We can break the differences of these approaches down into several sections: the number and effectiveness of the parameters we can give them, the general accuracy of each implementation, and finally the complexity of the computations.

It is possible for the Gabor filter to take several more parameters than the calculation of moments, this both increases the ability to fine tune each filter so that it provides a better segmentation result. However, with these options come more experimentation, and a greater amount of trial and error needed to achieve the best result. If some automation was done to find the best parameters to use, the complexity would be large. This means the moments approach was faster and easier to both implement and achieve the desired results. However, with that being said, the parameters that always made the most significant contribution to the program's output were the window sizes. Whether it was the size of the moment mask, the size of the Gabor filter, or the smoothing window after the activation function had been applied, these window sizes caused drastic changes in the results of the segmentation results.

In general, for these implementations, the Gabor filters performed better with images having higher ordered statistics. Tuceryan's algorithm was tested with several images having identical higher ordered statistics, and reliable results could not be gathered. This may be due to the unreliable input test images that were used. In Tuceryan's paper, only one of these types of images was segmented, with three clusters to account for boundary effects. At any rate, the Gabor filters performed these tasks with more accuracy. However, on other images, the accuracy was quite comparable, with all of the Brodatz texture images providing a good deal of accuracy for small spatial weights.

Lastly, the complexity of each approach is also comparable. If there were to be no conversions to the Frequency domain, then there would need to be two spatial convolutions performed for each approach, and one activation function used for each pixel. Without going into extraneous detail, we can observe that the more orientations and frequencies found for the Gabor filter the more feature images, and therefore more features per pixel, thus a higher complexity would occur. This is if we disregard the R^2 threshold. For the moments-based approach, if the $p + q$ threshold remains at 2, which it should, then there is no need to worry about a longer running time.

We have now compared both algorithms with respect to a problem known as textural segmentation, which is a very important problem with far reaching applications. These approaches are several years old, however, they are still quite effective solutions, and they offer some insight into how this task is handled, and the types of processing involved.

References

[1] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 3rd ed. Upper Saddle River, NJ: Pearson Prentice Hall, 2008, pp.827

[2] R. Shamey. Texture and Analysis of Texture: What is Texture? [Online]. Available: https://sites.textiles.ncsu.edu/color-science-lab/current-research/texture-and-analysis-of-texture. July 25, 2017.

[3] M. Tuceryan, "Moment-based texture segmentation," *Pattern Recognition Letters*, vol. 15, no. 7, pp. 659–668, 1994.

[4] A. Jain and F. Farrokhnia, "Unsupervised texture segmentation using Gabor filters," *1990 IEEE International Conference on Systems, Man, and Cybernetics Conference Proceedings*.

[5] J. Sklansky, "Image Segmentation and Feature Extraction," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 8, no. 4, pp. 237–247, 1978.

[6] H. Mobahi, S. R. Rao, A. Y. Yang, S. S. Sastry, and Y. Ma, "Segmentation of Natural Images by Texture and Boundary Compression," *International Journal of Computer Vision*, vol. 95, no. 1, pp. 86–98, Aug. 2011.

[7] D. Zhou, "Texture Analysis and its Applications," *Texture Analysis and its Applications*. [Online]. Available: https://www.cs.auckland.ac.nz/~georgy/research/texture/thesis-html/node7.html. [Accessed: 23-July-2017].

[8] M. Baatz and A. Schäpe, "Multiresolution segmentation: An optimization approach for high quality multi-scale image segmentation." *Angewandte Geographische Informationsverarbeitung*, vol. 12, no. 58, pp. 12–23, 2000.

[9] S. Mallick, "Blob Detection Using OpenCV," *Learn OpenCV*, 17-Feb-2015. [Online]. Available: https://www.learnopencv.com/blob-detection-using-opencv-python-c/. [Accessed: 21-July-2017].

[10] B. Julesz, E. N. Gilbert, L. A. Shepp, and H. L. Frisch, "Inability of Humans to Discriminate between Visual Textures That Agree in Second-Order Statistics—Revisited," *Perception*, vol. 2, no. 4, pp. 391–405, 1973.

[11] G. D. Boreman, *Modulation Transfer Function in Optical and Electro-Optical Systems*, *SPIE Press*, Bellingham, WA, 2001

[12] A. Bovik, *Handbook of Image Processing*, 2nd ed. Elsevier Academic Press, Burlington, MA: Elsevier Academic Press, 2005, pp. 1254s

[13] S.-C. Zhu, C.-E. Guo, Y. Wang, and Z. Xu, "What are Textons?," *International Journal of Computer Vision*, vol. 62, no. 1-2, pp. 121–143, 2005.

[14] J. Han and C. Moraga, "The influence of the sigmoid function parameters on the speed of backpropagation learning," *Lecture Notes in Computer Science From Natural to Artificial Neural Computation*, pp. 195–201, 1995.

[15] R. Owens, "Lecture 2", *Computer Vision IT412*, [Online]. Available: http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/OWENS/LECT2/node3.html. [Accessed: 23-July-2017].

[16] P. Brodatz*, Textures: A Photographic Album for Artists and Designers*, New York: Dover Publications, 1966

[17] K. Murthy, "Gabor Filters: A Practical Overview," *Computer Vision Tutorials*, 27-Apr-2014. [Online]. Available: https://cvtuts.wordpress.com/2014/04/27/gabor-filters-a-practical-overview/. [Accessed: 21-July-2017].

[18] G.N. Srinivasan and G. Shobha, "Statistical Texture Analysis," *PWASET*, vol. 36, pp. 1264-1269, Dec. 2008.