CS 696: Computer Vision
SDSU
12/18/2019

Ritu Gupta (RED ID: 824650002)
Garret Hughes (RED ID: 823446111)
Basile Hurat (RED ID: 813918996)

An Exploration of Empirical Wavelet Features Applied To Supervised Texture Segmentation

For this project, we explored many approaches to perform supervised texture segmentation.The main idea is to feed multiple machine learning algorithms with state of the art texture descriptors. The texture features are extracted by using Empirical Wavelet Transform. We build a unique empirical curvelet filter bank adapted to a given dictionary of textures. We then show that the output of these filters can be used to build efficient texture descriptors utilized to finally feed machine learning classifiers. Our approach is evaluated on a grayscale dataset and compared the results to various algorithms to show that the proposed methods perform comparably or better to naively implemented neural networks.

1. Introduction

In everyday life, people spend most of their time making decisions based on visual information that comes in many forms of varying shape, color, perspective, and texture. While each of these have their own challenges in computer vision, this paper will address the vast number of challenges that arise while processing and recognizing textures. Unlike color, texture is not described solely by pixel information. It does not have an ideal scale which best encapsulates the pattern, like shapes do. In addition, any blurring or noise can lead to large information loss with texture. As such, classic computer vision problems like supervised texture segmentation have historically been quite formidable.

However, recent work by Y. Huang and J. Gilles [4] has made great strides in solving these problems by using empirical wavelet features alongside sophisticated machine learning algorithms. The goal of this paper is to explore how well this cutting-edge feature extraction performs on supervised texture segmentation when using simpler machine learning algorithms.

1.1 Task Description

Supervised texture segmentation is the task of correctly identifying textures in a given image alongside a dataset of predetermined textures that are known to be present in an image. In rudimentary terms our task was accurately identifying a given region within an image based off of it's texture. We chose to use the Outex image database with black and white images. Outex is a framework for empirical evaluation of texture classification and segmentation algorithms [10]. Outex harps on a few key design principles: large versatile image database, a wide range of texture classification and segmentation problems, precise problem definition with test suites, trust, collaborative development and ongoing support. The collection of surface textures is expanding continuously. At this very moment the database contains 320 surface

textures, both macrotextures and microtextures [11]. These textures range from canvas to wood to gravel and a plethora of other common textures. While their dataset was exactly what we were looking for we ran into a few issues.

2. Major Challenges and Solutions

2.1 Challenges With The Task

We saw our first challenge when computing the first image. Since supervised texture segmentation is a pixel-wise classification, each image contains 65,536 (256x256) data point and then classified by 262,144 (512x512) separate variables. This posed a serious issue when we went to run SVM and kNN models. Given the large dataset we were comparing we were seeing times upward of three hours. To solve for SVM we first tried using the Parallel Toolkit to run batch jobs in parallel. While batch jobs seemed to help marginally, it turns out the fitcecoc method has an implemented parallel pool option that gives you 4 available workers on a single job. We considered using batch jobs within AWS using EC2 instances but fell short on time. I think this would be another solution to this problem.

Another challenge was an issue with the Outex dataset itself. For any given problem, it only includes the 5 textures that are in the image and for machine learning solutions, there is a serious lack of training data. For each composite image of textures, the folders have one image of each texture for training. Not only is this too small a set, it also has no interaction between the textures, and would likely result in a model that has poor performance on edges. As such, there was a need for us to apply some data augmentation. For the training stage, we generated an augmented training set of 100 folders with 250 images each from five pristine textures by randomly generating ground-truth images.These ground-truths are built via the following process: a) we create a 256 × 256 random image following a normal distribution within the range [0, 1]. b) we apply a Gaussian filtering ($\sigma = 20$), c) we binarize this filtered image using a threshold corresponding to the median value of the image. Then each connected components in that binary image are labeled to provide different regions (since we only have five pristine textures, we only keep the generated masks which provide five regions). Finally, each pristine texture is assigned to each mask region to obtain a training image.

2.2 Extracting Texture Relevant Features

As mentioned previously, texture segmentation has been a persistent open problem in computer vision. Much of that challenge stems from characterizing the textures themselves. Textures are nonlocal qualities that have incredible variability in their pixel-wise intensities. They are, instead, often categorized by their combination of frequencies and orientations. As such, feature designs related to textures often focus on that. Previously, local binary patterns (LBP), grey level co-occurrence matrices (GLCM), and Gabor filters have been used to extract texture features. Gabor filters, in particular, have been quite handy, as they are very flexible in extracting specific frequencies and orientations. Furthermore, the use of wavelets, which are an extension to gabor filters and are more flexible in their chosen basis functions, has proven to be quite useful.

In the last few decades, however, more adaptive methods have been discussed. One example is empirical mode decomposition, which is powerful and less rigid than the previously described methods. However, it is hard to generalize into 2D. This method also lacks mathematical foundations, which make it hard to interpret, generalize, and improve. However, in 2014, J.Gilles [1] proposed an adaptive wavelet transform to leverage the already useful Gabor filters and wavelets with the strength of an adaptive approach.

2.2.1 The Empirical Wavelet Transform

Since textures are composed primarily of orientations and frequencies, they often form significant harmonic modes in the fourier spectrum of an image. The empirical wavelet transform works, therefore, by attempting to separate those modes. While the details of the EWT can be found in papers put forth by J. Gilles in [1], [2], and [3], the basic idea is as such: given the modes of an image, we try to find the ideal separation between them at the lowest point between those two modes in the fourier spectrum. We then construct a filter bank from those separations and use the inverse fourier transform to extract empirical wavelet coefficients.

There are many ways to do this, and, for our project, we mirrored the approach used in [4], which was using the curvelet-1 option. Rather than trying to find an ideal 2D separation, it simplifies the image into two 1D signals by taking the pseudo-polar fourier transform, and then averaging the scales and angles. It then performs the 1D EWT on each to extract separations for the scales and for the orientations. Figure [1] shows an example of the Curvelet-1 empirical wavelet boundaries.

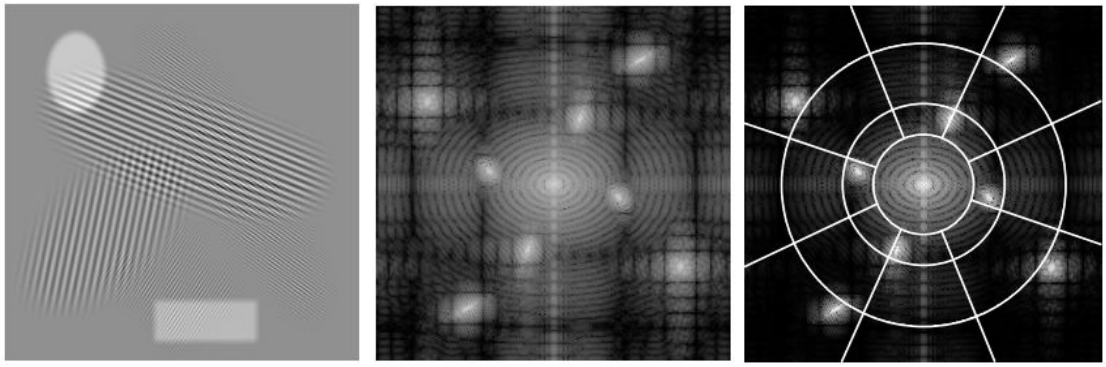


Fig. 1: A textured image, its fourier transform, and its Empirical Curvelet-1 boundaries

2.2.2 Empirical Wavelet Features

While the empirical wavelet transform is powerful enough to apply it to the supervised texture segmentation problem, we need to build one filter bank for each problem in order to have a consistent amount of features. To do this, we create a filter bank of the combined radial and angular boundaries of each training texture. However, some post-processing of our bounds is necessary to prevent having too many features. So, our approach is similar to the method used by the authors in [4].

1.) For each training texture, we extract the scalar and angular bounds and the maxima in each filter.
2.) We then combine the bounds and normalize them onto the range $[0,2\pi]$.

3.) For each pair of bounds, if there is not a maximum between them, we merge them at their midpoint.
4.) Then, based on a threshold (.2 for radial bounds, .07 for angular bounds), we remove bounds that are too close together, keeping the lower bound.

The result of this algorithm, along with examples of angular boundary detection of the training textures, is shown in Figure [2].

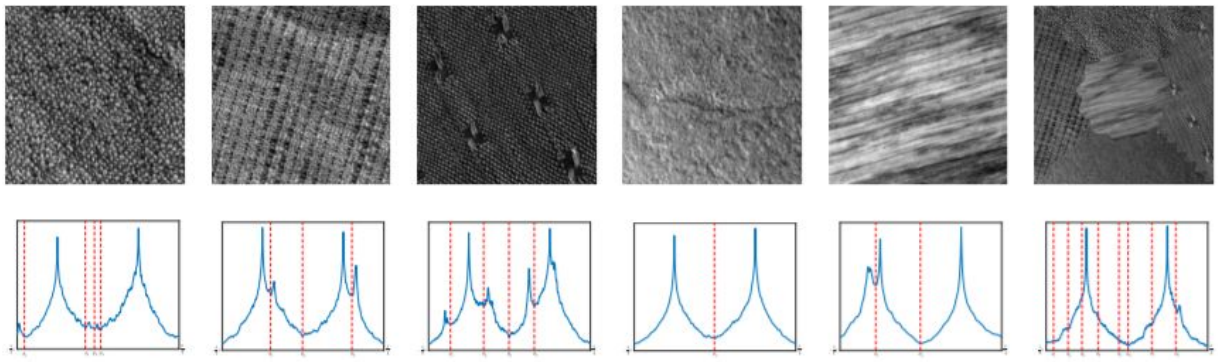


Fig. 2: Detection of angular boundaries for training textures and resulting merged boundary set

2.2.3 Pre- and Post-Processing

During the feature extraction process, we have both pre-processing of the images before the empirical wavelet feature extraction, and post-processing of those features. For pre-processing, we apply a cartoon-texture decomposition and keep only the texture component. This was shown to improve results with regards to unsupervised texture segmentation with k-means in [5], and, as k-NN and k-means are similar in approach, and since methods like SVM would also benefit, we took a similar approach here.

For post-processing, we perform a windowed local energy computation. A basic though nonlinear filter, local energy computation sums the squares of intensity values in a window around a pixel. Empirical wavelet features do not get rid of all of the variation in textures that cause problems, but, they mute the unrelated textures. Therefore, by adding the local energy around a pixel, we remove the pixel-wise variation while not adding extra information from the muted textures. The resulting feature is strictly positive, rather than oscillating, and represents the amount a texture of specific orientation and frequency is around a particular pixel. We tested multiple window sizes before determining an ideal size of 19x19.

3. Our choices for machine learning algorithms

For the past several years, computer vision solutions have been focused on one state of the art machine learning technique: Neural networks. This is no different for our problem, and many neural network architectures exist to model it. For comparison, we looked into the results of the FCNT, a fully connected network modified to do supervised texture segmentation by the authors in [7], and used alongside empirical wavelet features in [4]. The results of these neural networks will be discussed later, but, as we drafted our idea for this project, we wanted to highlight some two well known downsides of these types of models: Training time and dataset

size. Neural networks can take days, to weeks, to months to train correctly, and often require massive datasets to properly learn the features. As such, we chose our models in part based on training speed and scalability.

3.1 Naive Bayes Classifier

One of the first machine learning algorithms we considered was the naive Bayes classifier, which has many strengths. Firstly, it is easy to understand. Based on Bayes' theorem, it tries to find the class that maximizes $P(C|A_1, A_2, \ldots, A_n)$. By Bayes' theorem, however, this is equivalent to maximizing $P(A_1, A_2, \ldots, A_n|C)P(C)$, which, if assuming independence of features, becomes $P(C)\prod_{i=1}^{n} P(A_i|C)$. We notice that this is incredibly fast to calculate, which leads to the second major strength of the naive Bayes classifier: Its training and testing speed. Because NBC has an incredibly fast training and testing speed, we were able to use it to quickly tune the feature hyperparameters, for example. It also was the only machine learning algorithm that would train and test w/ the whole image set. As such, it was an important choice as a baseline machine learning algorithm for this project.

3.2 k-Nearest Neighbor

Our next choice was one of the simplest machine learning algorithms, k Nearest Neighbor, which stores all the available cases and classifies the new data or case based on a similarity measure. The 'k' in kNN algorithm is based on feature similarity and choosing the right value of k is important for better accuracy. To determine the values of the kNN hyperparameters, we experimented with smaller portions from training dataset to predict accuracy in cross-validation set using different values of k and distance metrics. For values of k from 1 to 3, k equal to 3 showed a reduction in the amount of noise in output segmented image. We considered 'cityblock' as distance metric over traditional 'euclidean' distance as it yielded better accuracy. We learned that kNN was simple and flexible for training image set but it was computationally very costly for predicting. To avoid high testing time as well as to maintain good predictability, we were able to train and test only 10 images per image folder.

3.3 Support Vector Machines

Support Vector Machine, otherwise known as SVM, is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples [8]. Since at its core, SVM is really just separation of classes we found that this would be an optimal approach toward extracting textures out of a given image. Once we implemented parallel pooling, we were able to achieve aggregate times consistently in the five to fifteen minute range. We decided this was the best approach and moved forward with parallel pooling with hyperparameters of Coding and CodingName of "all pairs". All pairs (also known as one-vs-one) uses a $K(K-1)/2$ number of binary learners. For each binary learner, one class is positive, another is negative, and the software ignores the rest. This design exhausts all combinations of

class pair assignments [9]. We found Coding to have the highest efficacy compared to its peers and once within Coding we found the allpairs (aka one-vs-one) value to yield the best results.

4. Experiment Results

For our experiments, we used varying numbers of training images based on the training/testing time of our machine learning models, which can be seen in Table 1.

Algorithm	Images used per folder	Folders tested on	Avg Testing time
Naive Bayes Classifier	250	0-99	17.57
k-Nearest Neighbor	10	0-99	2324.61
Support Vector Machines	10	0-99	41.92

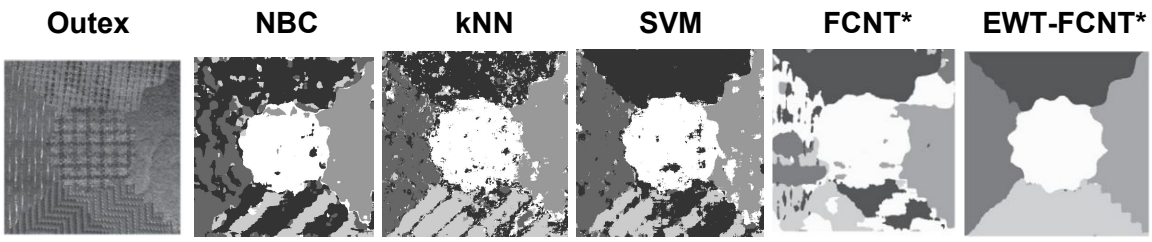
Table 1: Training depth for each method

Table 2 meanwhile shows the average result for various metrics. These metrics are commonly used to judge the accuracy of segmentation results and were the same one that the authors in [4] used to evaluate their models. We include their results in Table 2 for direct comparison. Also, Figure 3 shows examples of the segmentation results for each method.

Method	NVOI	SSC	SDHD	BGM	VD	BCE	Avg	Std. Dev
NBC	63.54	63.91	75.42	75.17	78.38	60.97	68.97	10.26
kNN	59.58	63.94	76.45	76.39	77.71	60.34	69.07	9.99
SVM	72.39	76.84	86.07	86.07	86.57	73.45	80.23	11.41
FCNT*	65.38	62.84	72.91	72.66	77.96	60.09	68.64	11.70
EWT-FCNT*	96.06	98.00	98.98	98.98	98.98	97.70	98.12	1.22

Table 2: Averaged metric results from each folder
***Results copied from Y.Huang & J.Gilles [4]**

Results



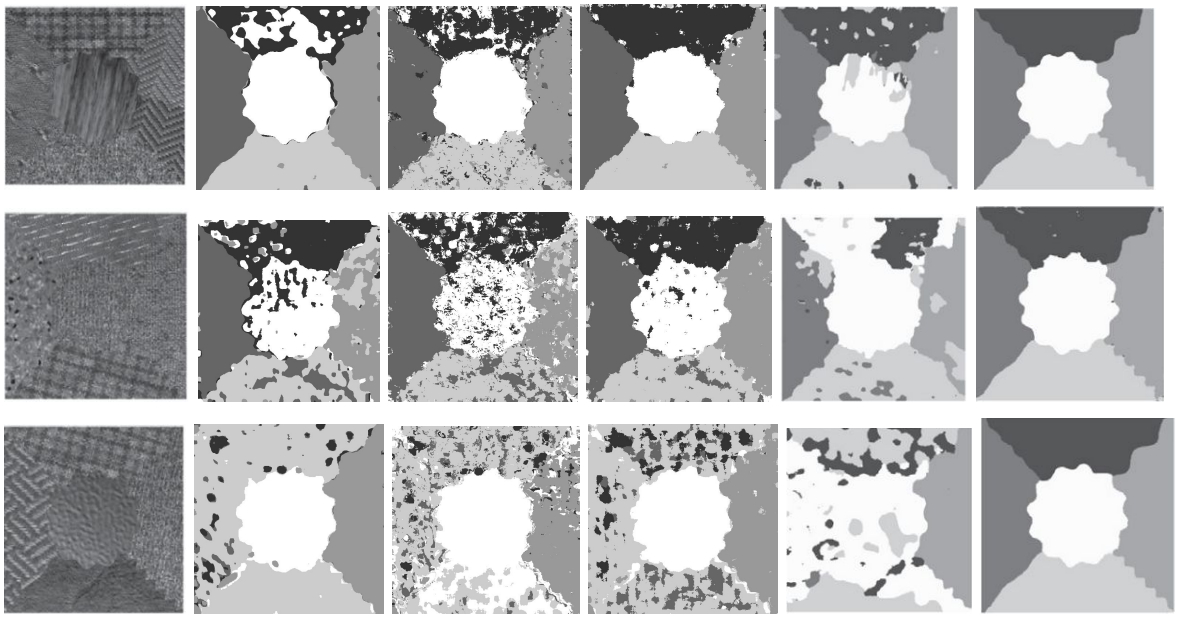


Fig. 3: Segmentation Results from Outex Image Set
*** Results copied from Y.Huang & J.Gilles [4]**

Naive Bayes Classifiers and k-Nearest Neighbor performed surprisingly well with these features, barely outperforming the FCNT network. Given the simplicity of both methods and the reasonable training and testing times, these appear to be decent alternatives to FCNT networks for grayscale texture segmentation. As far as NBC and kNN results are compared, kNN had slightly better prediction than NBC though it had large prediction time computation.

Support Vector Machine (SVM) performed the best for our tests. Due to the issues above around time limitations we ultimately made a decision to use SVM but in a dulled capacity. Instead of doing 100x250 images, we settled for 100x10 images. While that is significantly less than NBC went through, we're still confident that the delta in results would not change given the change in quantity. Ultimately we found that, outside of the empirical wavelet FCNT, SVM had the best overall accuracy as outlined in the results table above and while it didn't beat EWT-FCNT we did outperform just FCNT, which given the training time, is significant.

Finally, there are a few other small observations we can make. It can be seen that the three methods we attempted have similar failures when they fail, but those failures do not resemble those of the FCNT. The fourth image is a clear example of this, where our three methods each classify three regions as the bottom texture, but the FCNT has the center texture overrepresented. Also, there is a difference in the artifact shape of our methods. SVM and kNN have rather sharp edges to their artifacts and misclassifications, while NBC and FCNT are more rounded. Depending on the problem, some errors might be more preferable to others, especially if there are extra assumptions that might lead to some segmentation post-processing.

5. Conclusions and Future Works

In conclusion from our testing, it is clear that SVM performed the closest to FCNT. While only doing ten images per instead of 250 we were able to confidently show an average delta across the board. Our next best performer came in as kNN, then NBC as they barely outperformed the neural net with no features (68.64). Areas of improvement would be to first run the entire datasets through each method instead of cutting to ten for time. Secondly we, in the future, would like to test out each hyperparameter variation on the dataset, n * 250 *100 to confidently choose our hyperparameters instead of only off a select subset of the image folders and image counts. We learned that with well designed features you can outperform naive neural network implementations and save a lot of time and data acquisition. We found that these features are still worth designing as when they are fed into neural networks, improvements can be staggering.

Given that this is an exploratory project, there is still much more work to be done. Our tests were limited by time and computation power, but with proper investment, we could train kNN and SVM models on larger image sets, reducing possible variance and and possibly increasing accuracy of results. Also, there are other datasets of images, both grayscale and color, that would provide valuable insights on the performance of the empirical wavelet features when using NBC, kNN, and SVMs. Looking into the performance of the discussed methods on real applications would also be interesting.

Furthermore, there are further insights to be found in the improvement of features for each machine learning method. Since we used the same feature extraction for each of our methods, we were unable to see how data cleaning techniques, such as mean centering, normalizing, and proportional weighting might improve the results of each method individually and thus further work in those types of avenues could prove fruitful.

Bibliography

[1] Gilles, J. (2013). Empirical wavelet transform. IEEE Transactions on Signal Processing, 61(16), 3999–4010. https://doi.org/10.1109/TSP.2013.2265222
[2] Gilles, J., Tran, G., & Osher, S. (2014). 2D Empirical Transforms. Wavelets, Ridgelets and Curvelets revisited. xx, 1–31.
[3] Gilles, J., & Heal, K. (2014). A parameterless scale-space approach to find meaningful modes in histograms - Application to image and spectrum segmentation. 1–17.
[4] Huang, Y., Zhou, F., & Gilles, J. (2019). Empirical curvelet based fully convolutional network for supervised texture image segmentation. Neurocomputing. https://doi.org/10.1016/j.neucom.2019.04.021
[5] Huang, Y., De Bortoli, V., Zhou, F., & Gilles, J. (2018). Review of wavelet-based unsupervised texture segmentation, advantage of adaptive wavelets. IET Image Processing, 12(9), 1626–1638. https://doi.org/10.1049/iet-ipr.2017.1005
[6] Al-Kadi, O. (n.d.). Supervised Texture Segmentation: A Comparative Study.
[7] Andrearczyk, V., & Whelan, P. F. (2017). Texture segmentation with Fully Convolutional Networks. (March). Retrieved from http://arxiv.org/abs/1703.05230
[8] OpenCV docs. https://docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html
[9] Mathworks documentation https://www.mathworks.com/help/stats/fitcecoc.html
[10] Outex main documentation - http://www.outex.oulu.fi/
[11] Extended Outex test suites - http://lagis-vi.univ-lille1.fr/datasets/outex.html