CS 543: Final Project Report
Texture Classification using 2-D Noncausal HMMs

Felix Wang
fywang2

John Wieting
wieting2

1 Introduction

We implement a texture classification algorithm using 2-D Noncausal Hidden Markov Models. As with many of the various classification algorithms that were studied in the course, Hidden Markov Models (HMMs) assume an underlying structure to the object to be classified. Classifiers that follow this approach include the articulated parts models for classifying deformable objects, or the sliding window approach where a statistical template encodes the structure. However, unlike these classifiers, HMMs simply provide a basis on which the underlying structure of the object is to be estimated from observations as opposed to specifically defined. This doubly-stochastic approach has been used in areas such as speech recognition and video event classification where the process to be measured is well modeled by both a hidden and an observed component [1][2].

Transferring this methodology to the task of estimating and classifying textures, we impose some modifications from the standard HMM left-to-right causal structure to accommodate the temporal stationarity of a 2-D image. That is, we move from the standard understanding of state transitions to the idea of state dependencies. The organization of the report is as follows: in Section 2, we present the algorithm in detail and its performance for a sample dataset; in Section 3, we discuss the results of the experiment with respect to related work; we conclude in Section 4.

2 Experiment

For our texture dataset, we use 8 texture classes taken from the Ponce research group at UIUC [3]. This dataset contains a total of 25 texture classes with 40 intensity images per class of size 640×480 pixels. Due to this size, we trimmed the outer edges of the images we trained and tested on to 256×256 pixels. The classes that we used from this dataset were: bark, wood, water, granite, pebbles, brick, fur, and knit. A set of representative images containing these textures can be seen in Figure 2.

Our classifier can be split into a learning component where we estimate the parameters of the HMM for the training images, and a classification component where we compute the likelihood of a test image. Out of these 40 images, we partitioned them into 10 for learning model parameters and 30 for classification. The basic structure of our learning algorithm is taken from the work by Povlow and Dunn in their work on texture classifications using a similar technique [4]. Here, modifications are made such that the observation space for a given state is continuous as opposed to discrete. The learning algorithm is further decomposed into an initialization step, and an estimation-maximization step. For the actual classification of a test images, after computing a likelihood with respect to the parameters from training, we assign the texture class as the one whose learned parameters which produced the highest likelihood. This is essentially a nearest neighbor approach to classification, with the likelihood function substituting as our distance.

Figure 1: Representative images of the texture classes using in our classifier

The model we assumed for the texture images is composed of two states representing a Gaussian distribution over pixel intensities. That is, each state defines a mean and standard deviation of pixel intensities for the pixels that belong to that state. This defines the observation probabilities given a state and pixel intensity.

$$p_q(\xi_i) \equiv p(\text{observation } \xi_i | q) \tag{1}$$

The other component of the model considers the state dependencies of our hidden states. Here the dependency of the state of any given pixel is a function of its four neighbors (left, above, right, below). The 2-D property of our model is a result of this spatial dependency, as is the noncausality. The entries into our state dependency matrix can then be formulated as:

$$P_{q|r,s,t,u}(\lambda_{m,n}) \equiv P\left(\lambda_{m,n} = q | \lambda_{m,n-1} = r, \lambda_{m-1,n} = s, \lambda_{m,n+1} = t, \lambda_{m+1,n} = u\right) \tag{2}$$

where $\lambda_{m,n} = q$ is the state at the pixel location m, n, and similarly with its neighbors.

During the initialization of our model for the learning phase of our algorithm, we assign the mean pixel intensity of the two states to the $\frac{1}{3}$ and $\frac{2}{3}$ quantiles of the pixel intensity distribution of the image we wish to learn. The standard deviation is chosen such that the means of the two states are two standard deviations apart from each other. Although this scheme is chosen for the initial state parameters, we initialize the assignment of states to pixels as random so that the initial estimated state dependency matrix is more or less uniform. A depiction of this initial representation of state assignments can be seen in the middle image of Figure 2 which gives an illustration of our learning pipeline for a sample image.

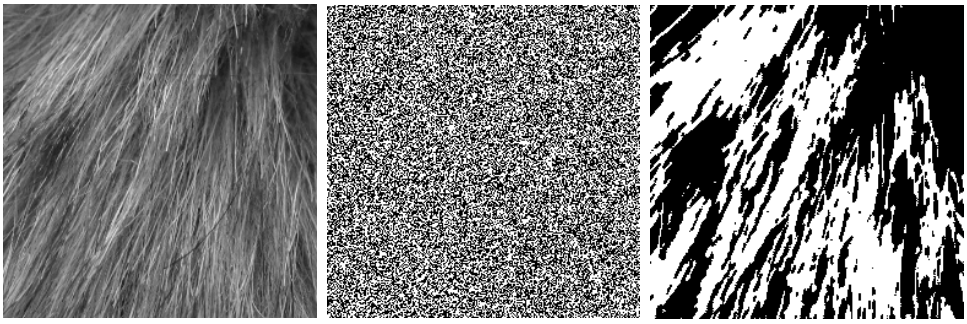

Figure 2: Learning pipeline (from left to right): original texture, initial states, and final states

In the estimation step, we recompute the parameter to the model. For the state dependencies, we simply perform counting over all the possible distribution of a state and its neighbors. Due to the relatively small amount of edge pixels that do not have all these neighbor states, we choose to ignore them in the reestimation.

$$P_{q|r,s,t,u} = \frac{\sum_{m=2}^{M-1}\sum_{n=2}^{N-1} I[q,r,s,t,u]}{\sum_{m=2}^{M-1}\sum_{n=2}^{N-1} I[r,s,t,u]} \tag{3}$$

Here, $I[\cdot]$ is an indicator function and the states q, r, s, t, u take on two values each, giving a total of 32 entries in our state dependency matrix. In reestimating the observation probabilities, we use the standard Gaussian probability density function

$$p_q(\xi_i) = \frac{1}{\sigma_q^2\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{\xi_i - \mu_q}{\sigma_q}\right)^2} \tag{4}$$

where μ_q and σ_q are calculated by simply taking the mean and standard deviation, respectively, of the pixel intensities assigned to state q.

In the maximization step, we calculate the local likelihood of the a pixel over all possible states given its neighbors and choose the state that gives the highest likelihood. This is arguably the most computationally expensive step as we update the state distribution for all the pixels in the image by calculating several local likelihood functions per pixel.

$$L(\Lambda', \mathcal{X}) = P_{q'|r,s,t,u} P_{r|a,b,q',h} P_{s|b,c,d,q'} P_{t|q',d,e,f} P_{u|h,q',f,g} p_{q'}(x_{m,n}). \tag{5}$$

Where q' is the modified state at location m, n, Λ represents the states and $\mathcal{X}$ the intensities. The other indices in the arguments to the state dependency probabilities refer to the arrangement of states shown in Figure 2. The reason to take a local likelihood function is that the modification of a single state only affects the values in its immediate vicinity; the computed likelihood of the remaining locations of the image remains unchanged and can thus be ignored.

		c		
	b	s	d	
a	r	q	t	e
	h	u	f	
		g		

Figure 3: Neighbor designation for local likelihood function

These two steps, estimation and maximization, are iteratively performed until the convergence criteria is met. In our case, we simply took the convergence criteria to be 5 iterations as the convergence rate for such HMM based methods are relatively fast. Alternatively we could compare to some tolerance value the change in the likelihood of the image given the updated parameters in an iteration to a previous one, or perhaps terminate when the state of the pixels no longer change. The resulting state distribution along with the learned parameters for it are then returned by the learning algorithm. For our sample image, the final state distribution after learning is shown in the rightmost image in Figure 2.

Classification is performed by computing a likelihood of a model learned for a test image in terms of the parameters learned over the training images. In terms of the model parameters, this likelihood function can be computed as

$$L(\Lambda, \mathcal{X}) = \prod_{m=2}^{M-1}\prod_{n=2}^{N-1} P(\lambda_{m,n}|\lambda_{m,n-1}, \lambda_{m+1,n}, \lambda_{m,n+1}, \lambda_{m-1,n})\, p_{\lambda_{m,n}}(x_{m,n}) \tag{6}$$

Again, due to the minor contributions of the edge pixels, they can be ignored. From the calculated likelihoods, then, the classification of an image to a texture was performed by matching the parameters that contributed to the largest likelihood to the texture class of the training image that generated those parameters. By this method, we attained an overall classification accuracy for our 8 texture classes of 76.25%. The confusion matrix for this is shown in Figure 2. The indices of the confusion matrix correspond, in order, to the textures that were presented in Figure 2 from left-to-right and then top-to-down.

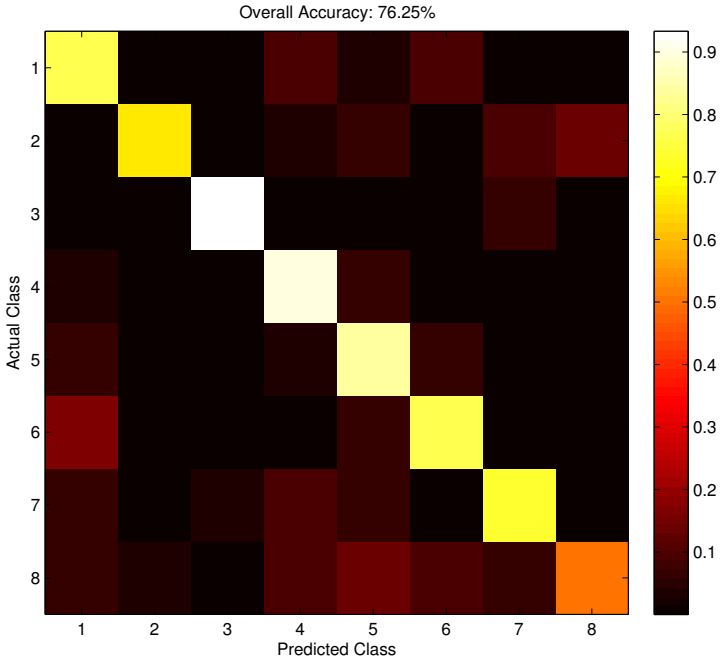

Figure 4: Confusion Matrix for 2-D Noncausal HMM Texture Classification

3 Discussion

In their paper [4], Povlow et al. implement a discretized observation space version of the algorithm presented here, and they achieve classification accuracies of 95.8%, 89.1%, and 92.7%. Respectively, the datasets that they performed classification on were: synthetic textures, generated using a method proposed by Challepa, textures from the Bordatz dataset, and those consisting of images photographed by the authors themselves. All images were 256×256 pixels in size, which the authors reduced to a set of 16 subimages of size 64×64 pixels. Each image had a 6-bit quantization of pixel intensities (64 possible levels), and unlike the images used for our experiment, there was little variation among images belonging to a texture class. This was primarily due to extraction method of subsampling a larger image, whereas the images in the Ponce dataset belonging to a texture class were taken from not only different locations and illumination conditions, but there was also variation with respect to scale and orientation [3]. For reference, the classification algorithm associated with the authors of our dataset was performed using a combination of detector/descriptor pairs such as Harris/RIFT or Laplacian/Spin, they achieved an overall accuracy of 92.61% by using information from all detector/descriptor pairs over 10 training images.

Due to the additional complexity and variation of the Ponce dataset as compared to those classified by Povlow et al., we can not simply average the parameters derived from the training step for a texture class. In their implementation, this averaging is precisely what was performed by Povlow et al. as it allowed for a reduction of minor noise during their training. For our dataset however, this additional step during training led to the loss of valuable information tied to image properties such as orientation. When we implemented this additional step, the classification accuracy, 62%, was much less than when nearest neighbor was performed over all training images individually. Of course, the increase in accuracy comes at a computational cost, with the computation of the likelihood function over 80 sets of parameters for a test image taking roughly 7 seconds. It is possible that our approach could be more scalable if we clustered our images beforehand so we would not need to do a comparison with each image in our database individually. We could also introduce in our model parameters that would essentially accommodate for the variances in the dataset. Revisions to the classification algorithm could also be made such that instead of simply taking the most likely class, 1-NN, the mode of the $k > 0$ neighbors may give a better result.

4 Conclusion

In this report, we described a 2D noncausal Hidden Markov Model approach to texture classification. Using the Ponce dataset for our textures, we were able to achieve a classification accuracy of 76.25% using this approach. These results show that, although not at the current state of the art, this approach is a viable one for texture classification, especially if modifications to the algorithm are made such that variations between texture samples are accounted for.

References

[1] S. E. Levinson, *Mathematical Models for Speech Technology*. Wiley, 2005.

[2] X. Ma, D. Schonfeld, and A. Khokhar, "Video event classification and image segmentation based on noncausal multidimensional hidden markov models," *IEEE Transactions on Image Processing*, vol. 18, pp. 1304–1313, June 2009.

[3] S. Lazebnik, C. Schmid, and J. Ponce, "A sparse texture representation using local affine regions," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, pp. 1265–1278, August 2005.

[4] B. Povlow and S. Dunn, "Texture classification using noncausal hidden markov models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, pp. 1010–1014, October 1995.