Texture Synthesis Guided Deep Hashing for Texture Image Retrieval

Ayan Kumar Bhunia[1] Perla Sai Raj Kishore[2] Pranay Mukherjee[2] Abhirup Das[2] Partha Pratim Roy[3]
[1]Nanyang Technological University, Singapore
[2] Institute of Engineering & Management, India [3] Indian Institute of Technology Roorkee, India
[1]ayanbhunia@ntu.edu.sg

Abstract

With the large scale explosion of images and videos over the internet, efficient hashing methods have been developed to facilitate memory and time efficient retrieval of similar images. However, none of the existing works use hashing to address texture image retrieval mostly because of the lack of sufficiently large texture image databases. Our work addresses this problem by developing a novel deep learning architecture that generates binary hash codes for input texture images. For this, we first pre-train a Texture Synthesis Network (TSN) which takes a texture patch as input and outputs an enlarged view of the texture by injecting newer texture content. Thus it signifies that the TSN encodes the learnt texture specific information in its intermediate layers. In the next stage, a second network gathers the multi-scale feature representations from the TSN's intermediate layers using channel-wise attention, combines them in a progressive manner to a dense continuous representation which is finally converted into a binary hash code with the help of individual and pairwise label information. The new enlarged texture patches from the TSN also help in data augmentation to alleviate the problem of insufficient texture data and are used to train the second stage of the network. Experiments on three public texture image retrieval datasets indicate the superiority of our texture synthesis guided hashing approach over existing state-of-the-art methods.

1. Introduction

Recent times have seen a huge explosion of digital images over the internet which has made large image databases prevalent. Given any database one might want to search for images semantically using a query image. Content Based Image retrieval (CBIR) explored in [20, 23, 32, 35, 37, 38] provides a solution to the above mentioned problem by retrieving a set of similar images by the measure of a similarity metric between the feature representations of the query image and the member images of the database. Thus a general image retrieval pipeline consists of two steps : first, characterization of each image by rich discriminative features and second, performing a similarity search by some metric using these features to retrieve similar images.

Feature representations for images are usually continuous-valued and thus running a nearest neighbour search on these representations for retrieval turns out to be very slow and computationally inefficient, especially for real-time applications in mobile devices or in databases with millions of images. Hence, there is a need to optimize this naive search technique under the constraints of both space and time. Hashing [47] is one such state-of-the-art technique for Approximate Nearest Neighbor (ANN) search [17, 34, 36] used due to faster retrieval speeds and reduced memory footprint. It involves learning a hash function for encoding continuous-valued image descriptors to compact binary codes while preserving their similarity and discriminative properties. The similarity between two such hash codes can be easily computed by their Hamming distance with the simple XOR operation. Recently, performance of traditional hashing methods have been bettered by deep hashing techniques like [4, 5, 13, 21, 22, 31, 54, 55] which employ deep neural networks for learning the hash function. Consequently, our work uses deep hashing to address a special kind of image retrieval task called Texture Image Retrieval.

Texture image retrieval may be defined as a type of CBIR which aims at searching for images having texture-patterns semantically similar to that of the query image. Texture being a low level visual attribute of an image's surface acts as a representative of the surface's roughness and also provides useful visual cues about the object's identity. Texture image retrieval has a variety of applications such as in digital library, multimedia web search, multimedia storage system and query-based video investigation. However, this task is very challenging for a couple of reasons. Firstly, due to the lack of large scale texture databases, this task is much less explored in the context of deep learning as compared to ordinary image retrieval. Secondly, for image retrieval tasks in general, the closeness among images is usually governed by the similarity among their high level features. However, in texture image retrieval where similarity is governed by texture patterns, similar semantics become difficult to capture owing to the fact that texture is a low level visual attribute.

In our work, we introduce a deep hashing framework for texture image retrieval guided by a Texture Synthesis Network (TSN) [52]. We extract information at various levels from the intermediate layers of a pre-trained TSN and combine these multi-scale activations using channel-wise attention in a progressive manner to generate a powerful set of feature descriptors for texture images. The example-based texture synthesis approach aims at generating a texture image, double the size of input that faithfully captures all the visual properties of the input by preserving the large-scale structural features, the natural appearance and the spatial variation of local patterns. For this we use a generative adversarial network (GAN) where the generator aims to synthesize a larger image with an expanded view of the given input texture and the discriminator aims at comparing and classifying the generated textures with the corresponding ground truth images. Although this might resemble image super-resolution [24], they are different in various ways. Image super-resolution simply aims at enhancing the quality of the given image by increasing its resolution and sharpening it, whereas, texture synthesis aims at expanding the view of the given texture patch by injecting additional content consistent with the given input. Since the generator is able to synthesize larger textures from smaller patches, it is evident that all of the texture related information is recorded in the intermediate layers of the generator network. This information is used as the key source in our work for generating good descriptors for texture images.

Channel-wise attention helps us to combine activation maps from various layers of the pre-trained TSN in a selective manner and reduce noise, such that channels with more information are given greater weightage than noisy and less informative ones. These continuous valued features are finally hashed to generate compact binary coded representations of the images. In addition to images from existent texture datasets, we also make use of images generated from our TSN to alleviate the problem of insufficient texture data for training which significantly improves the performance of our deep neural network.

In this paper, we make the following novel contributions:

- We introduce a deep hashing network for texture image retrieval, which uses the rich texture information recorded in the intermediate layers of a pre-trained TSN, filters them using channel-wise attention and then combines these multi-scale feature representations in a progressive manner to get a powerful and robust set of feature descriptors for texture images. We finally learn a hash function to map these continuous valued feature descriptors to dense binary codes.
- To the best of our knowledge, our work is the first to introduce an end-to-end deep neural network and hashing for texture image retrieval. Experimental results on various benchmark datasets exhibit a superior performance of our framework over existing methods.

2. Related Work

Earlier methods for image retrieval relied on encoding techniques [15, 40, 41] for aggregating local patches to build a global image representation. Later methods such as [11, 39] relied on convolutional neural networks (CNNs) for learning image features. Krizhevsky *et al*. [18] used feature information gathered by a classifier, pre-trained on large scale dataset like ImageNet, as the basis of developing suitable descriptors for instance-level retrieval tasks. A more recent work by Radenovic *et al*. [43] aims at generating good descriptors for image retrieval by fine-tuning CNNs on a large collection of unordered images in a fully automated manner. The authors of [11] proposed a large-scale dataset which they cleaned to produce less noisy training data for image retrieval. Furthermore, they improved the R-MAC feature descriptor [46] and used a triplet loss for training it using a siamese network.

Few of the existing works compressed the descriptors to improve the storage requirements and retrieval efficiency at the cost of reduced accuracy. Both supervised [12] and unsupervised [14, 41, 42] compression techniques are used for this purpose. The issue of compressing the feature descriptors for faster retrieval without compromising on accuracy is solved by hashing. Traditional hashing methods largely differ from the state-of-the-art deep learning based methods which learn the image representations and hash codes in an end-to-end manner.

The authors of [51] proposed one of the first approaches to deep hashing called Convolutional Neural Network Hashing (CNNH) which used a two stage network for learning the image representations and the hash codes. However, their hash code learning process did not receive any kind of feedback from the learnt image representations. Network in Network Hashing (NINH) [21] attempts to alleviate the problem faced in CNNH by learning the image representations and the hash codes at the same stage. Many other ranking-based deep hashing methods [48, 53, 55] and pairwise label based deep hashing methods [29, 57] have been proposed in the recent years. The hashing part of our work draws inspiration from [28] which proposes a single framework to learn the binary codes directly by using both pairwise and individual label information.

Although hashing has been proved to significantly increase the time and memory efficiency, it has still not been used in texture image retrieval. Earlier methods on texture image retrieval relied on handcrafted features like Discrete wavelet transform, Gabor features [26], local binary

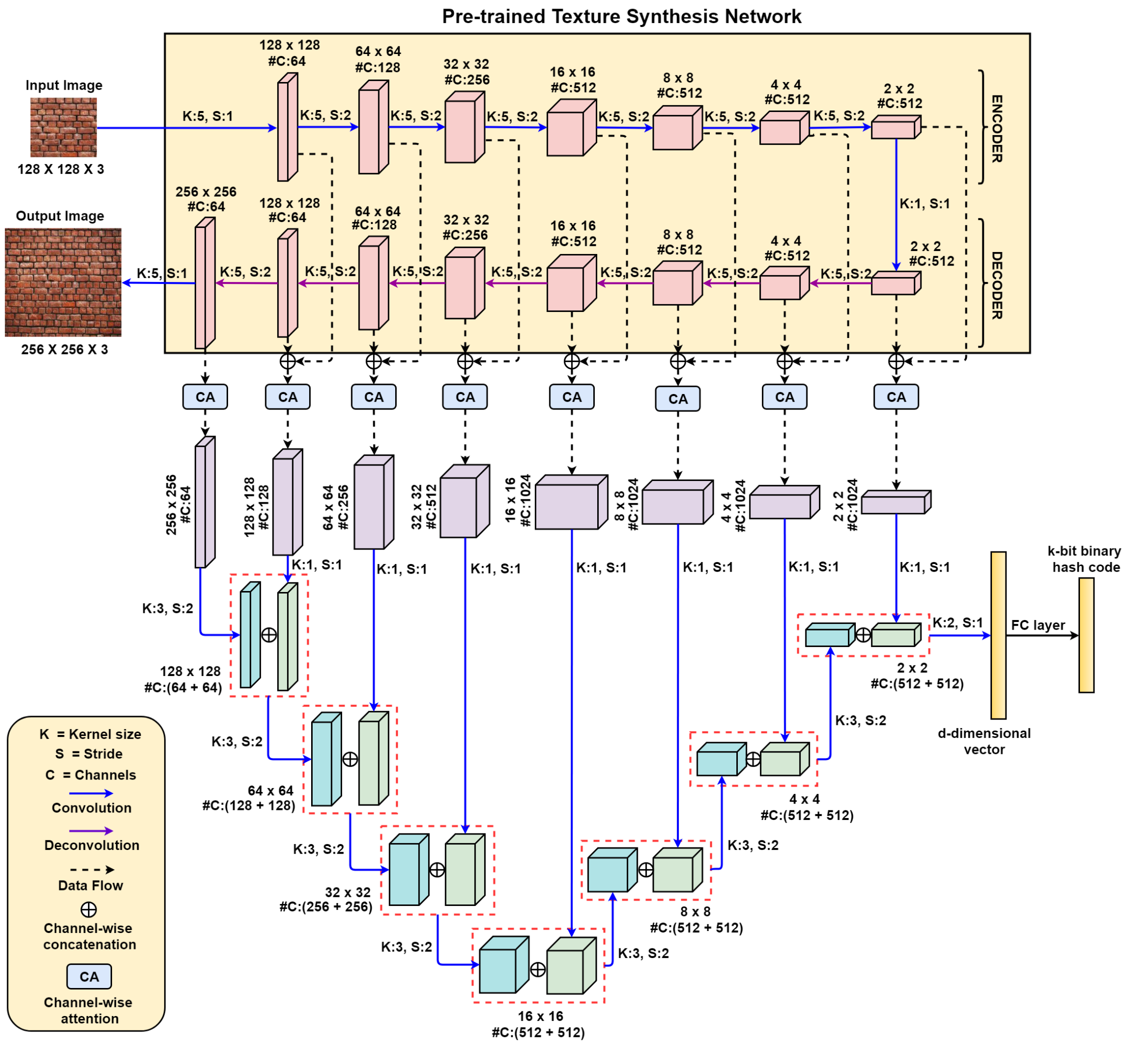

Figure 1: A 128 × 128 texture patch is passed as input to the pre-trained generator of the TSN which generates an expanded texture patch of size 256 × 256 as output. The corresponding activation maps of the same size from the encoder and decoder part of the generator are depth-wise concatenated and are passed through a channel-wise attention (CA) module to generate a filtered version of the intermediate features, followed by a 1 × 1 convolution operation. We use separate CA modules for every pair of feature combination between encoder and decoder side of activation maps. The multi-scale features are then combined in a progressive manner using successive downsampling (by strided convolution) and depth-wise concatenation.

pattern (LBP) [33], rotated wavelet filter, rotated complex wavelet filter and dual tree complex wavelet transform for feature extraction. Although LBP worked well for grayscale images, it performed poorly for colour images. To alleviate this problem, the authors of [33] introduced color-information feature (CIF) in addition to LBP to generate textural and color information for images under the settings of image retrieval and classification. Li *et al.* [26] use Gabor wavelets to decompose colour texture images into various dependencies, analyse them and then capture these dependencies with Gaussian copula models. A recent work by Banerjee *et al.* [2] introduces a new feature descriptor called Local Neighbourhood Intensity Pattern (LNIP). We argue that in order to learn good feature descriptors for texture image retrieval, it is necessary to understand the underlying patterns and the low-level and high-level properties of the texture images. This can be done effectively by learning to generate a larger texture image from a smaller patch by injecting newer patterns in a consistent and non-repeating manner.

Texture synthesis aims exactly at this task and has been extensively researched for over the past two decades. Classical methods for texture synthesis include non-parametric approaches, such as [6, 19, 25, 49, 50]. Whereas recent approaches [3, 16, 27, 52, 56] make use of deep learning based methods to significantly improve the quality of results. One of the earliest methods for texture synthesis using deep neural networks include the work of Gatys *et al*. [8] where they iteratively optimize an image to generate texture. Fang *et al*. [7] proposed a novel application of texture synthesis for editing images by applying texture to the surface of a photographed object. TextureGAN [52] uses a deep generative network that aims to synthesize textures in sketch images guided by user defined texture samples. In [56], Zhou *et al*. proposed a state-of-the-art generative approach for doubling the spatial extent of the given input patch by injecting newer texture content.

3. Proposed Work

This work introduces a novel deep hashing framework for texture image retrieval. We aim to generate good feature descriptors for texture images by transferring knowledge learned from the task of texture synthesis to texture image retrieval. Our framework can be divided into two stages: In the first stage, we train the TSN using an adversarial setup, where the generator learns to generate large texture images from smaller patches by injecting newer texture content. In the second stage, the pre-trained TSN is used for feature extraction of texture images for hashing. We combine the multi-scale activation maps from the intermediate layers of pre-trained generator network in a selective and efficient manner to generate a powerful set of feature descriptors. Finally, these continuous valued feature descriptors are hashed to compact binary vectors for efficient memory usage and faster retrieval. These binary valued vectors are then compared by their Hamming distance using a simple XOR operation. We also tackle the data insufficiency problem for training the deep network by using the newly generated texture patches from the TSN for learning the hash function. The details of our model and the training procedure is discussed in the subsequent sections.

3.1. Texture Synthesis Network

Generating large texture images from smaller patches forms the basis of generating powerful feature descriptors for the texture images in our work. A generative adversarial approach is employed for texture synthesis, with a convolutional encoder-decoder network as the generator and a fully convolutional classification network as the discriminator.

The generator $G_{TSN}(.)$ takes in a smaller texture patch of size $K \times K$ as input and generates a larger texture image of size $2K \times 2K$ via adversarial expansion. In the encoder part of the generator, after every strided convolution operation, the spatial dimensions of the activation maps are halved and the number of channels are increased. Similarly in the decoder part, after each deconvolution operation, the spatial dimensions of the activation maps are doubled and the number of channels are reduced. The generator network is by purpose designed in a symmetric manner to allow easy combination of activation maps from the corresponding layers of the encoder and decoder network in the second stage of our framework. It is also to be noted that the generator is not perfectly symmetrical due to the extra deconvolution operation in the decoder part which is necessary for upsampling the image to size $2K \times 2K$. The details of the generator architecture has been shown in Figure 1. The discriminator, $D_{TSN}(.)$, is a traditional convolutional classification network which takes in a large texture image (of size $2K \times 2K$) and classifies it as being generated by the generator (fake) or not (real). Our discriminator has an architecture similar to the one used in [44].

Along with the adversarial loss L_{adv} [10], for simultaneous training of the generator and the discriminator, we follow [56] to use two additional loss terms which guide the generator to generate high quality expanded textures over time. We use L_1 loss between the image patch output by the generator and it's ground truth counterpart from the examples so as to provide a pixel-wise error supervision to the generator. Second, we use style loss L_{style} [9] to enforce perceptual similarity between the generated image and the ground truth image. For this, we use Gram matrices [9] computed from the activations of convolutional layers of the pre-trained VGG-19 network. Therefore, the final loss equation is as follows:

$$L_{total} = L_{adv} + \gamma_1 L_{style} + \gamma_2 L_{L_1} \qquad (1)$$

The pre-trained generator of this TSN is used for two purposes: first, to extract powerful feature descriptors for the texture images from its intermediate layers and second, to generate more data to train the second stage of the network for hashing.

3.2. Channel-wise Attention

In the second stage of our framework, we perform channel-wise concatenation of the corresponding activation maps of the same size from the encoder and decoder parts of the pre-trained TSN. The reason behind this concatenation is that we intend to combine both local and global information from the intermediate layers of the TSN. The activations from the encoder part hold the local texture-specific information extracted from the input patch, whereas the activations from the decoder part hold information about the inherent properties of the texture and a global class-specific representation of the same. Since the decoder part is mainly responsible for generating an expanded view of the input texture by injecting additional texture content, it can be interpreted that the information contained in the activation maps of the decoder part is more representative of the inherent global properties of the texture image. Every channel of any convolutional feature map has high response to some specific content of the image. Thereby it becomes important to assign more priority to those channels which contain more discriminative information in order to facilitate better feature learning. The Channel-wise Attention (CA) module in our framework is mainly used to encode this characteristic and in the process provides higher weightage to the channels carrying more texture specific information than the channels with lesser information. The output of the CA module is a tensor of same dimensions as that of input but channels weighted as per their scalar relevance value.

An intermediate output of the CA module is a vector A_c having dimension equal to the number of input channels C, along which the attention operation is applied. Here, every i^{th} element of A_c represents the normalized contribution of the corresponding i^{th} channel to the output of the attention module. A_c is calculated as follows :

$$A_c = softmax(W_c * q + b) \tag{2}$$

In the above equation, $*$ operator represents the convolution operation, W_c represents the convolutional filter, b represents the bias factor and q represents the vector of dimension C obtained by performing global average pooling on every channel of the input to the channel-wise attention module. This normalized vector is finally multiplied with the original input tensor to get a filtered output. We use separate CA modules for every pair of feature combination between the encoder and the decoder side of activation maps. In our case, we use 8 different CA modules (see Figure 1) in order to adaptively weigh different channels of the combined feature maps of sizes ranging from 256×256 till 2×2 by powers of 2.

3.3. Progressive Multi-scale Feature Combination

Following the channel-wise attention operation, we intend to combine these multi-scale feature maps to get a final feature vector representation of the input texture patch. This progressive multi-scale feature combination module includes two operations: first, a 1×1 convolution operation upon the filtered output of the CA module and second, progressive combination through successive downsampling by strided convolution and depth-wise concatenation. There are two main reasons behind the use of 1×1 convolution: feature mixing by combining channel-wise weighted feature maps and dimensionality reduction by reducing the number of channels equal to the depth of the previous layer for concatenation at a later stage. Let CA^M represent the output of a CA module applied to feature map of size $M \times M$, where M has values 256, 128, 64, 32, 16, 8, 4, and 2 . 1×1 convolution is applied to every CA^M , except where M equals to 256, resulting in a feature map $F^M_{1\times1}$. First, CA^{256} is passed through a strided convolutional layer giving $F^{128}_{strided}$ and then concatenated with $F^{128}_{1\times1}$ which is of same depth. In a similar fashion, the concatenated feature map is again passed through another strided convolutional layer by outputing $F^{64}_{strided}$ and similar operations are performed for successive downsampled feature maps to finally generate a d dimensional vector. Therefore, it can be generalized from $M = 128$ onwards as follows:

$$F^M_{1\times1} = Conv_{1\times1}(\text{CA}^M) \tag{3}$$

$$F^{M/2}_{strided} = Conv_{strided}(F^M_{strided} \oplus F^M_{1\times1}) \tag{4}$$

where, $Conv_{strided}(\cdot)$ is a strided convolution layer with kernel size 3×3 and stride 2, and $Conv_{1\times1}(\cdot)$ is a 1×1 convolutional layer with number of filter equals to half of the depth of its input. A clearer and vivid visual description can be found in Figure 1.

3.4. Hash Function

The previous sections involved generating a d dimension feature descriptor for a given input texture patch. This section describes the use of hashing which converts these continuous feature representations to dense binary codes while preserving the semantic similarity between the images.

In the problem of hashing, given a set of N images each of which have been described by the features of dimension d the aim is to encode the feature matrix, Z, of dimension $d \times N$ into another matrix H of dimension $k \times N$ such that every element of H is either 1 or -1. In other words every column of Z that represents the image's complex features in d dimensions is mapped to a binary feature representation of dimension k. Thus if we consider $h(.)$ to be the hashing function, this process can be explained mathematically as $H_i = h(Z_i)$, where H_i represents the i^{th} column of matrix H and Z_i represents the i^{th} column of matrix Z.

Following [28], we make use of both pairwise label information and individual label information for supervising the hashing network. The similarity measure between two binary hash codes (b_i, b_j) is defined using the concept of Hamming distance $dist_h(\cdot,\cdot)$ and cosine similarity $\langle\cdot,\cdot\rangle$ as $dist_h(b_i, b_j) = \frac{1}{2}(k - \langle b_i, b_j\rangle)$. Since the inner product and Hamming distance are inversely related, we can use inner product to quantify similarity between the hash codes. Negative log likelihood reduces the Hamming distance between similar pair of images and increases the Hamming distance between dissimilar pairs. A simple linear classification network is used along side to exploit the label information directly and is based on the assumption that the learned binary codes should be good enough for classification as well. Therefore, our final loss function is a weighted summation of negative log likelihood function J [28] and classification loss Q [28], and is given by :

$$L_{hash} = J + \nu Q \tag{5}$$

Since optimizing L_{hash} is a discrete optimization problem, we follow the training and testing procedure proposed in [28] for this purpose.

4. Experiments

4.1. Datasets

To demonstrate the efficiency of our method, we evaluated the proposed texture synthesis guided hashing framework on three popular texture datasets by comparing it against several state-of-the-art hashing frameworks. We use the **MIT-VisTeX** (full) database which is a collection of 167 unique texture patterns of size 512×512 with one image belonging to every class. Next, we also show our experimental results on the Salzburg Texture **(STex)** database which is a collection of 476 unique texture patterns recorded under real-world conditions. Every image of this database is of size 1024×1024 with one image belonging to every category of texture. We select this database to test the performance of our framework for higher number of classes. We also show our experimental evaluations on the Amsterdam Library of Textures **(ALOT)** database that contains coloured images of 250 unique texture patterns. For every unique texture pattern, 100 images were recorded by varying various scientific parameters. Apart from this, linear mixtures of 12 materials are also included in the database totalling upto more than 27500 images in the database. This database was chosen because it contains a larger intra-class variance which necessitates a robust framework for handling the same.

4.2. Implementation details

Data Preparation: Since both MIT-Vistex and STex contain only one image for every texture class, we resize all the images from these datasets to 1024×1024 and divide every image into 4 regions. Now randomly sampled patches covering three-fourth of the image region are used for training and rest one-fourth is used for sampling test patches, so that every testing sample remains unseen to the trained model. On the other side, ALOT has 100 samples for every texture class and we randomly select 70 images for training, 20 for testing and rest 10 for validation. This split is maintained for every experiment and we denote this division as $\{D_{train}, D_{test}, D_{val}\}$. For our experiments, D_{val} comes only from the ALOT dataset and all the hyperparameters are tuned based on this and remain constant for all the experiments. In our framework, there are two stages of training: we first train the Texture Synthesis Network (TSN) in an adversarial setup alongwith a discriminator using the objective in Equation (1). Thereafter, we use the pre-trained TSN as a multi-scale feature extractor and train the second stage of the network with the objective in Equation (5) that finally outputs a k bit binary vector for a given input texture patch. Following this, we use two types of training samples for training these two stages. *Stage - 1*: To train the TSN, we use paired data $\{I^{128}_{inp,stage1}, I^{256}_{gt,stage1}\}$, where $I^{256}_{gt,stage1}$ is an image of size 256×256 randomly sampled during every iteration of *Stage - 1* from D_{train} and $I^{128}_{inp,stage1}$ is a randomly cropped 128×128 texture patch contained inside the $I^{256}_{gt,stage1}$. *Stage - 2*: As the second stage is responsible for finally generating the binary hash codes, the second stage is supervised with the paired data $\{I^{i}_{inp,stage2}, L^{i}_{gt,stage2}\}^{N}_{i=1}$, where $I^{i}_{inp,stage2}$ is an image of size 128×128 cropped randomly from D_{train} and $L^{i}_{gt,stage2}$ denotes the corresponding class label of i^{th} training sample. For every experiment, we generate a total of 2000 texture patches (from D_{train}) with respect to each class in order to train the network. During inference, we randomly generate 200 texture images (from D_{test}) of size 128×128 from every class which are used as query to evaluate the performance.

Training: *Stage - 1*: During the first stage of training, the generator $G_{TSN}(\cdot)$ takes $I^{128}_{inp,stage1}$ as input and outputs I^{256}_{G} of size 256×256. The discriminator, $D_{TSN}(\cdot)$, tries to classify between the generated image I^{256}_{G} (fake) and the corresponding ground truth $I^{256}_{gt,stage1}$ (real), and gives rise to adversarial loss, L_{adv}. Reconstruction loss (L_{L_1}) is calculated by taking the absolute difference between I^{256}_{G} and $I^{256}_{gt,stage1}$. We use a VGG-19 model pretrained on Imagenet to calculate the style loss (L_{style}) using Gram matrices computed at the output of the following layers : $relu1\_1, relu2\_1, relu3\_1, relu4\_1$ and $relu5\_1$. Following [8], a weighted summation of the corresponding losses is done by setting the weights to $0.244, 0.061, 0.15, 0.004, 0.004$ respectively. A Gaussian distribution of mean 0 and standard deviation 0.02 is used to initialize the weights and biases of all the the convolutional layers. Following [56], we use the Adam optimizer with initial learning rate 0.0002 and set momentum to 0.5. We train the network for 100000 epochs, and γ_1 and γ_2 are set to 100 and 1 respectively. Once the training of *Stage - 1* is complete, we freeze all the weights of $G_{TSN}(\cdot)$ and use it as a feature extractor. *Stage - 2*: In the second stage, we feed $I^{i}_{inp,stage2}$ to the TSN (i.e. $G_{TSN}(\cdot)$) and extract multi-scale feature representations from its intermediate layers. To alleviate the problem of insufficient data, we also use the generated texture patches I^{256}_{G} from the output of TSN by sampling random patches of size 128×128, in order to train the second stage of network in an end-to-end manner. This aids in data augmentation and helps the second stage of the network to generalize well even from limited amount of original data. While training the second stage of the network, we follow the procedure used by the authors of [28].

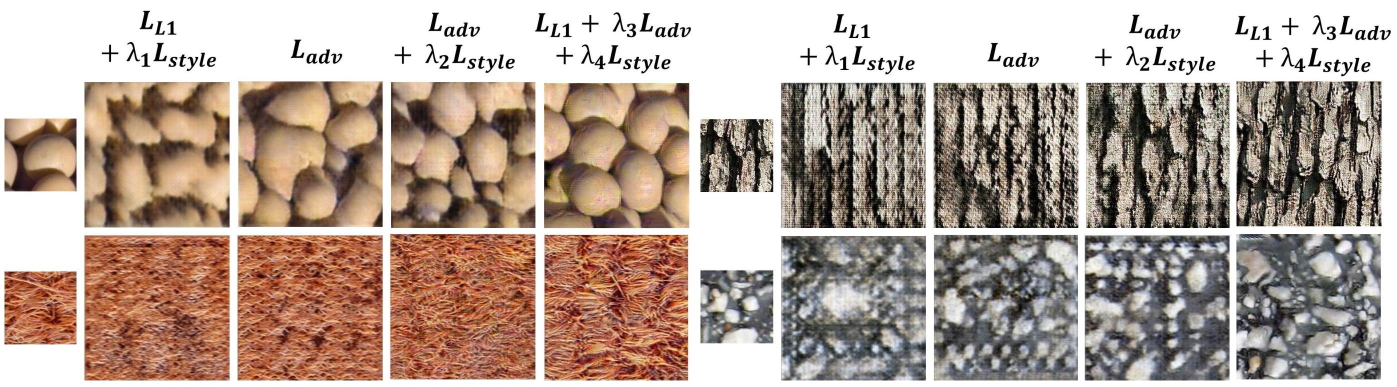
$L_{L1} + \lambda_1 L_{style}$
L_{adv}
$L_{adv} + \lambda_2 L_{style}$
$L_{L1} + \lambda_3 L_{adv} + \lambda_4 L_{style}$

Figure 2: Images generated by the Texture Synthesis Network. The smaller patches denote the inputs to the Synthesis Network and the remaining columns show results generated by using different combinations of loss functions.

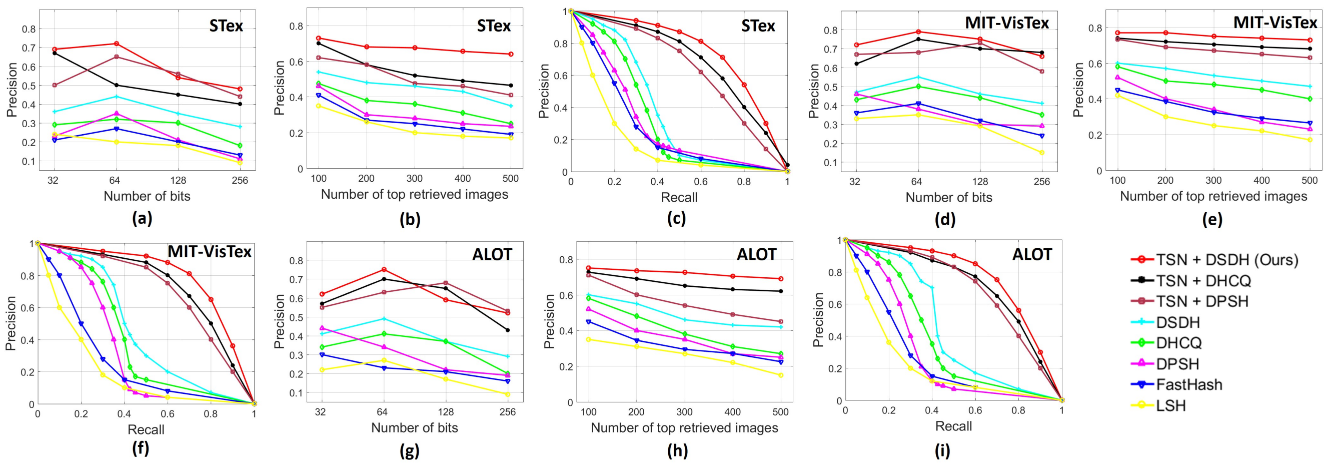

Figure 3: (a)-(c), (d)-(f), (g)-(i) show precision (Hamming radius ≤ 2) vs. number of bits, precision vs. number of top retrieved images, and precision vs. recall curve for STex, MIT-VisTex, and ALOT datasets respectively.

4.3. Performance Analysis

In recent years, there has been a number of works addressing different loss functions to train end-to-end networks for hashing. DSDH [28] employs a deep CNN-F architecture and uses pairwise label and classification information for generating hash codes. Similar to DSDH, DPSH [29] uses pairwise label information, but learns feature representations and hash codes simultaneously. Similar to DPSH, DHCQ [45] goes for simultaneous learning of feature representations and hash codes based on classification and quantization errors. Due to superior performance on various benchmark datasets, we have used the loss functions and training methods proposed in DSDH [28] to train the second stage of our framework which finally outputs a k-bit binary vector. However, our generative model guided deep hashing framework is a meta-framework and can be easily extended with any other state-of-the-art's objective functions for training. Therefore, we have used objective functions from two other end-to-end trainable deep hashing frameworks [29, 45] in order to justify the efficacy of our framework. We name the experimental setups as TSN-DSDH, TSN-DPSH, TSN-DHCQ; where we train the second phase of our network, guided by a generative model, using the loss functions introduced in DSDH, DPSH and DHCQ respectively. We also compare with the original frameworks used in DSDH, DPSH, DHCQ which are broadly based on feed-forward convolutional neural network. We also compare against traditional hashing methods like FastHash [30] and LSH [1] where every texture patch is represented by a standard LBP feature descriptor. Following other works [28, 45], we evaluate Mean Average Precision (MAP) for different methods using same standard split-

Methods	STex				VisTex				ALOT			
	32 bits	64 bits	128 bits	256 bits	32 bits	64 bits	128 bits	256 bits	32 bits	64 bits	128 bits	256 bits
TSN + DSDH (Ours)	**0.60**	**0.624**	**0.646**	**0.655**	**0.731**	**0.746**	**0.759**	**0.762**	**0.711**	**0.722**	**0.737**	**0.741**
TSN + DHCQ	0.598	0.617	0.631	0.643	0.682	0.702	0.719	0.734	0.688	0.7	0.712	0.721
TSN + DPSH	0.587	0.611	0.629	0.632	0.667	0.699	0.711	0.717	0.64	0.661	0.675	0.688
DSDH [28]	0.44	0.464	0.475	0.482	0.554	0.582	0.6	0.606	0.518	0.54	0.555	0.56
DHCQ [45]	0.432	0.46	0.471	0.478	0.542	0.559	0.573	0.581	0.451	0.49	0.527	0.542
DPSH [29]	0.430	0.420	0.415	0.413	0.498	0.512	0.52	0.524	0.43	0.458	0.483	0.498
FastHash [30]	0.369	0.381	0.394	0.396	0.387	0.414	0.431	0.449	0.324	0.354	0.386	0.41
LSH [1]	0.347	0.363	0.377	0.382	0.368	0.395	0.411	0.419	0.303	0.351	0.37	0.376

Table 1: MAP values of different methods using top-500 retrieved images.

Variants		STex	VisTex	ALOT
TSN Loss	$L_{L1} + L_{style}$	0.19	0.23	0.213
	L_{adv}	0.482	0.566	0.551
	$L_{adv} + L_{style}$	0.586	0.671	0.643
	$L_{adv} + L_{style} + L_{L1}$	0.624	0.696	0.722
No CA		0.587	0.651	0.678
No Data Augmentation		0.556	0.592	0.659

Table 2: MAP values for different variants of the proposed model using code length of 64 bits.

ting protocol on three different datasets (see section 4.2). Table 1 depicts the MAP values at different lengths of hash code. In Figure 3, we have shown the plot for precision vs. top-T retrieved samples, precision vs. number of bits in hash code and precision vs. recall keeping the threshold for Hamming distance equal to 2. From Figure 3, it is evident that the performance of DSDH, DPSH, DHCQ is limited. Whereas, the same loss objective significantly boosts up the performance when combined with our TSN guided framework. Comprehensive experiments also validate that our method is the most time efficient with the average retrieval time per query (in seconds) being 0.00104, 0.00169 and 0.00281 for ALOT, MIT-VisTeX and STeX respectively. From our observation, this significant improvement is mainly due to two reasons: first, our generative model guided framework can facilitate better feature learning, and second, the generated outputs of TSN help in data augmentation for the training of second stage of our framework.

4.4. Ablation Study

In our framework, the central idea is to use a powerful generative model (TSN) to help generate better feature representations for texture image retrieval. Therefore, we carry out a comprehensive study to judge how image retrieval performance is related to the quality of image generation network. Figure 2 depicts the quality of generated texture patches due to the use of different combinations of loss functions. In four different experiments, we train the TSN using the following combinations of the loss functions : a) L_1 loss and style loss (b) adversarial loss (c) adversarial and style loss (d) adversarial loss, style loss and L_1 loss. Thereafter, we use such pre-trained TSN networks as feature extractors in the second stage of training one at a time. We notice the performance due to pre-trained TSN using combination (a) to be very poor. The main driving force of this TSN network is the adversarial loss function. However, adding both L1 Loss and style loss alongside helps the TSN to generate better quality images, and thereby signifying a significant rise in the texture retrieval performance. The performance due to different possible combinations of loss objectives to train the TSN is shown in Table 2. This implies that the better the generated images are, the more powerful are the feature representations preserved in the intermediate layers, thereby significantly improving the retrieval performance. Apart from considering the importance of individual losses of TSN, we also conduct experiments to verify the importance of channel-wise attention. We notice a drop of 0.037, 0.045 and 0.044 in the MAP values (with code length 64) for VisTex, STex and ALOT dataset respectively by removing the channel-wise attention in the second stage. We also conduct experiments without using data augmentation from the generated texture patches. Consequently, we notice a drop in the performance which is significantly large for Vistex and STex datasets due to them having less amount of data.

5. Conclusion

We have introduced a novel deep hashing architecture for texture image retrieval. Our framework first pre-trains a TSN which learns to synthesize an expanded view of a given texture thus recording texture specific information in its intermediate layers. The binarized hash codes are finally obtained by gathering feature maps from these intermediate layers and combining them in a selective and progressive manner. We also alleviate the problem of limited training data by using the generated texture patches from the TSN for training. Thus we conclude that, the idea of extracting more robust features guided by generative networks can further be extended to other image retrieval problems.

References

[1] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *FOCS*, pages 459–468, 2006. 7, 8

[2] P. Banerjee, A. K. Bhunia, A. Bhattacharyya, P. P. Roy, and S. Murala. Local neighborhood intensity pattern–a new texture feature descriptor for image retrieval. *Expert Systems with Applications*, 113:100–115, 2018. 3

[3] U. Bergmann, N. Jetchev, and R. Vollgraf. Learning texture manifolds with the periodic spatial gan. *arXiv preprint arXiv:1705.06566*, 2017. 4

[4] F. Cakir, K. He, and S. Sclaroff. Hashing with binary matrix pursuit. In V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, editors, *ECCV*, pages 344–361, 2018. 1

[5] Z. Cao, M. Long, J. Wang, and S. Y. Philip. Hashnet: Deep learning to hash by continuation. In *ICCV*, pages 5609–5618, 2017. 1

[6] A. A. Efros and T. K. Leung. Texture synthesis by non-parametric sampling. In *ICCV*, page 1033, 1999. 4

[7] H. Fang and J. C. Hart. Textureshop: texture synthesis as a photograph editing tool. In *ACM Transactions on Graphics*, volume 23, pages 354–359, 2004. 4

[8] L. Gatys, A. S. Ecker, and M. Bethge. Texture synthesis using convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 262–270, 2015. 4, 6

[9] L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. In *CVPR*, pages 2414–2423, 2016. 4

[10] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014. 4

[11] A. Gordo, J. Almazan, J. Revaud, and D. Larlus. End-to-end learning of deep visual representations for image retrieval. *International Journal of Computer Vision*, 124(2):237–254, 2017. 2

[12] A. Gordoa, J. A. Rodriguez-Serrano, F. Perronnin, and E. Valveny. Leveraging category-level labels for instance-level image retrieval. In *CVPR*, pages 3045–3052, 2012. 2

[13] K. He, F. Cakir, S. A. Bargal, and S. Sclaroff. Hashing as tie-aware learning to rank. *arXiv preprint arXiv:1705.08562*, 2017. 1

[14] H. Jégou and O. Chum. Negative evidences and co-occurences in image retrieval: The benefit of pca and whitening. In *ECCV*, pages 774–787. 2012. 2

[15] H. Jégou, M. Douze, C. Schmid, and P. Pérez. Aggregating local descriptors into a compact image representation. In *CVPR*, pages 3304–3311, 2010. 2

[16] N. Jetchev, U. Bergmann, and R. Vollgraf. Texture synthesis with spatial generative adversarial networks. *arXiv preprint arXiv:1611.08207*, 2016. 4

[17] Y. Kalantidis and Y. Avrithis. Locally optimized product quantization for approximate nearest neighbor search. In *CVPR*, pages 2329–2336, 2014. 1

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 2

[19] V. Kwatra, A. Schödl, I. Essa, G. Turk, and A. Bobick. Graphcut textures: image and video synthesis using graph cuts. *ACM Transactions on Graphics*, 22(3):277–286, 2003. 4

[20] R. Kwitt and A. Uhl. Image similarity measurement by kullback-leibler divergences between complex wavelet subband statistics for texture retrieval. In *ICIP*, pages 933–936, 2008. 1

[21] H. Lai, Y. Pan, Y. Liu, and S. Yan. Simultaneous feature learning and hash coding with deep neural networks. In *CVPR*, pages 3270–3278, 2015. 1, 2

[22] H. Lai, P. Yan, X. Shu, Y. Wei, and S. Yan. Instance-aware hashing for multi-label image retrieval. *IEEE Transactions on Image Processing*, 25(6):2469–2479, 2016. 1

[23] N.-E. Lasmar and Y. Berthoumieu. Gaussian copula multivariate modeling for texture image retrieval using wavelet transforms. *IEEE Transactions on Image Processing*, 23(5):2246–2261, 2014. 1

[24] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. P. Aitken, A. Tejani, J. Totz, Z. Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *CVPR*, volume 2, page 4, 2017. 2

[25] S. Lefebvre and H. Hoppe. Appearance-space texture synthesis. In *ACM Transactions on Graphics*, volume 25, pages 541–548, 2006. 4

[26] C. Li, Y. Huang, and L. Zhu. Color texture image retrieval based on gaussian copula models of gabor wavelets. *Pattern Recognition*, 64:118–129, 2017. 2, 3

[27] C. Li and M. Wand. Precomputed real-time texture synthesis with markovian generative adversarial networks. In *ECCV*, pages 702–716, 2016. 4

[28] Q. Li, Z. Sun, R. He, and T. Tan. Deep supervised discrete hashing. In *Advances in Neural Information Processing Systems*, pages 2482–2491, 2017. 2, 5, 6, 7, 8

[29] W.-J. Li, S. Wang, and W.-C. Kang. Feature learning based deep supervised hashing with pairwise labels. *arXiv preprint arXiv:1511.03855*, 2015. 2, 7, 8

[30] G. Lin, C. Shen, and A. van den Hengel. Supervised hashing using graph cuts and boosted decision trees. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 37(11):2317–2331, 2015. 7, 8

[31] K. Lin, H.-F. Yang, J.-H. Hsiao, and C.-S. Chen. Deep learning of binary hash codes for fast image retrieval. In *CVPR*, pages 27–35, 2015. 1

[32] G.-H. Liu, Z.-Y. Li, L. Zhang, and Y. Xu. Image retrieval based on micro-structure descriptor. *Pattern Recognition*, 44(9):2123–2133, 2011. 1

[33] P. Liu, J.-M. Guo, K. Chamnongthai, and H. Prasetyo. Fusion of color histogram and lbp-based features for texture image retrieval and classification. *Information Sciences*, 390:95–111, 2017. 3

[34] F. Magliani, T. Fontanini, and A. Prati. Efficient nearest neighbors search for large-scale landmark recognition. *arXiv preprint arXiv:1806.05946*, 2018. 1

[35] F. Magliani and A. Prati. An accurate retrieval through r-mac+ descriptors for landmark recognition. In *Proceedings of the 12th International Conference on Distributed Smart Cameras*, pages 6:1–6:6. ACM, 2018. 1

[36] M. Muja and D. G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, pages 2227–2240, 2014. 1

[37] S. Murala, R. Maheshwari, and R. Balasubramanian. Local tetra patterns: a new feature descriptor for content-based image retrieval. *IEEE Transactions on Image Processing*, 21(5):2874–2886, 2012. 1

[38] T. Ojala, M. Pietikäinen, and D. Harwood. A comparative study of texture measures with classification based on featured distributions. *Pattern recognition*, 29(1):51–59, 1996. 1

[39] M. Paulin, J. Mairal, M. Douze, Z. Harchaoui, F. Perronnin, and C. Schmid. Convolutional patch representations for image retrieval: an unsupervised approach. *International Journal of Computer Vision*, 121(1):149–168, 2017. 2

[40] F. Perronnin and C. Dance. Fisher kernels on visual vocabularies for image categorization. In *CVPR*, pages 1–8, 2007. 2

[41] F. Perronnin, Y. Liu, J. Sánchez, and H. Poirier. Large-scale image retrieval with compressed fisher vectors. In *CVPR*, pages 3384–3391, 2010. 2

[42] F. Radenovic, H. Jegou, and O. Chum. Multiple measurements and joint dimensionality reduction for large scale image search with short vectors-extended version. *arXiv preprint arXiv:1504.03285*, 2015. 2

[43] F. Radenović, G. Tolias, and O. Chum. Fine-tuning cnn image retrieval with no human annotation. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 2018. 2

[44] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015. 4

[45] J. Tang, Z. Li, and X. Zhu. Supervised deep hashing for scalable face image retrieval. *Pattern Recognition*, 75:25–32, 2018. 7, 8

[46] G. Tolias, R. Sicre, and H. Jégou. Particular object retrieval with integral max-pooling of cnn activations. *arXiv preprint arXiv:1511.05879*, 2015. 2

[47] J. Wang, H. T. Shen, J. Song, and J. Ji. Hashing for similarity search: A survey. *arXiv preprint arXiv:1408.2927*, 2014. 1

[48] X. Wang, Y. Shi, and K. M. Kitani. Deep supervised hashing with triplet labels. In *ACCV*, pages 70–84, 2016. 2

[49] L.-Y. Wei and M. Levoy. Fast texture synthesis using tree-structured vector quantization. In *Annual conference on Computer graphics and interactive techniques*, pages 479–488, 2000. 4

[50] Y. Wexler, E. Shechtman, and M. Irani. Space-time completion of video. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, pages 463–476, 2007. 4

[51] R. Xia, Y. Pan, H. Lai, C. Liu, and S. Yan. Supervised hashing for image retrieval via image representation learning. In *AAAI*, volume 1, page 2, 2014. 2

[52] W. Xian, P. Sangkloy, J. Lu, C. Fang, F. Yu, and J. Hays. Texturegan: Controlling deep image synthesis with texture patches. *arXiv preprint*, 2017. 2, 4

[53] T. Yao, F. Long, T. Mei, and Y. Rui. Deep semantic-preserving and ranking-based hashing for image retrieval. In *IJCAI*, pages 3931–3937, 2016. 2

[54] R. Zhang, L. Lin, R. Zhang, W. Zuo, and L. Zhang. Bit-scalable deep hashing with regularized similarity learning for image retrieval and person re-identification. *IEEE Transactions on Image Processing*, 24(12):4766–4779, 2015. 1

[55] F. Zhao, Y. Huang, L. Wang, and T. Tan. Deep semantic ranking based hashing for multi-label image retrieval. In *CVPR*, pages 1556–1564, 2015. 1, 2

[56] Y. Zhou, Z. Zhu, X. Bai, D. Lischinski, D. Cohen-Or, and H. Huang. Non-stationary texture synthesis by adversarial expansion. *arXiv preprint arXiv:1805.04487*, 2018. 4, 6

[57] H. Zhu, M. Long, J. Wang, and Y. Cao. Deep hashing network for efficient similarity retrieval. In *AAAI*, pages 2415–2421, 2016. 2