Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/pr

RADAM: Texture recognition through randomized aggregated encoding of deep activation maps

Leonardo Scabini[a,b,*], Kallil M. Zielinski[a], Lucas C. Ribas[c], Wesley N. Gonçalves[d], Bernard De Baets[b], Odemir M. Bruno[a,*]

[a] São Carlos Institute of Physics, University of São Paulo, São Carlos, postal code 13566-590, SP, Brazil
[b] KERMIT, Department of Data Analysis and Mathematical Modelling, Ghent University, Coupure links 653, Ghent, postal code 9000, Belgium
[c] Institute of Biosciences, Humanities and Exact Sciences, São Paulo State University, São José do Rio Preto, postal code 15054-000, SP, Brazil
[d] Faculty of Computing, Federal University of Mato Grosso do Sul, Campo Grande, postal code 79070-900, MS, Brazil

A R T I C L E I N F O

Article history:
Received 24 March 2023
Revised 16 June 2023
Accepted 2 July 2023
Available online 3 July 2023

Keywords:
Texture analysis
Randomized neural networks
Transfer learning
Convolutional networks
Feature extraction

A B S T R A C T

Texture analysis is a classical yet challenging task in computer vision for which deep neural networks are actively being applied. Most approaches are based on building feature aggregation modules around a pre-trained backbone and then fine-tuning the new architecture on specific texture recognition tasks. Here we propose a new method named **R**andom encoding of **A**ggregated **D**eep **A**ctivation **M**aps (RADAM) which extracts rich texture representations without ever changing the backbone. The technique consists of encoding the output at different depths of a pre-trained deep convolutional network using a Randomized Autoencoder (RAE). The RAE is trained locally to each image using a closed-form solution, and its decoder weights are used to compose a 1-dimensional texture representation that is fed into a linear SVM. This means that no fine-tuning or backpropagation is needed for the backbone. We explore RADAM on several texture benchmarks and achieve state-of-the-art results with different computational budgets. Our results suggest that pre-trained backbones may not require additional fine-tuning for texture recognition if their learned representations are better encoded.

1. Introduction

For several decades, texture has been studied in Computer Vision as a fundamental visual cue for image recognition in several applications. Despite lacking a widely accepted theoretical definition, we all have developed an intuition for textures by analyzing the world around us from material surfaces in our daily life, through microscopic images, and even through macroscopic images from telescopes and remote sensing. In digital images, one abstract definition is that texture elements emerge from the local intensity constancy and/or variations of pixels producing spatial patterns roughly independently at different scales [1].

The classical approaches to texture recognition focus on the mathematical description of the textural patterns, considering properties such as statistics [2], frequency [3], complexity/fractality [4,5], and others [6]. Many such aspects of texture are challenging to model even in controlled imaging scenarios. Moreover, the wild nature of digital images also results in additional variability, making the task even more complex in real-world applications.

Recently, the power of deep neural networks has been extended to texture analysis by taking advantage of models pre-trained on big natural image datasets. These transfer-learning approaches combine the general vision capabilities of pre-trained models with dedicated techniques to capture additional texture information, achieving state-of-the-art performance on several texture recognition tasks [7–9]. Therefore, most of the recent works on deep texture recognition propose to build new modules around a pre-trained deep network (backbone) and to retrain the new architecture for a specific texture analysis task. However, even if the new modules are relatively cheap in terms of computational complexity, resulting in good inference efficiency, the retraining of the backbone itself is usually costly. Going in a different direction, Randomized Neural Networks [10–12] proposes a closed-form solution for training neural networks, instead of the common backpropagation, with various potential applications. For instance, the training time of randomization-based models was analyzed [13] on datasets such as MNIST, resulting in gains up to 150 times. These gains can be expressive when hundreds of thousands of images are used to train a model.

* Corresponding authors at: São Carlos Institute of Physics, University of São Paulo, postal code 13566-590, São Carlos, SP, Brazil.
E-mail addresses: scabini@ifsc.usp.br (L. Scabini), bruno@ifsc.usp.br (O.M. Bruno).

https://doi.org/10.1016/j.patcog.2023.109802
0031-3203/© 2023 Elsevier Ltd. All rights reserved.

In this work, we propose a new module for texture feature extraction from pre-trained deep convolutional neural networks (DCNNs). The method, called **R**andom encoding of **A**ggregated **D**eep **A**ctivation **M**aps (RADAM), goes in a different direction than recent literature on deep texture recognition. Instead of increasing the complexity of the backbone and then retraining everything, we propose a simple codification of the backbone features using a new randomized module. The method is based on aggregating deep activation maps from different depths of a pre-trained convolutional network, and then training Randomized Autoencoders (RAEs) in a pixel-wise fashion for each image, using a closed-form solution. This module outputs the decoder weights from the learned RAEs, which are used as a 1-dimensional image representation. This approach is simple and does not require hyperparameter tuning or backpropagation training. Instead, we propose to attach a linear SVM at the top of our features, which can be simply used with standard parameters. Our code is open and is available in a public repository.[1] In summary, our main contributions are:

(i) We propose the RADAM texture feature encoding technique applied over a pre-trained DCNN backbone and coupled with a simple linear SVM. The model achieves impressive classification performance without needing to fine-tune the backbone, in contrast to what has been proposed in previous works.
(ii) Bigger backbones and better pre-training improve the performance of RADAM considerably, suggesting that our approach scales well.

2. Background

We start by conducting a literature review on texture analysis with deep learning and randomized neural networks.

2.1. Texture analysis with deep neural networks

In this work, we focus on transfer-learning-based texture analysis by taking advantage of pre-trained deep neural networks. For a more comprehensive review of different approaches to texture analysis, the reader may consult [14]. There have been numerous studies involving deep learning for texture recognition, and here we review them according to two approaches: feature extraction or end-to-end fine-tuning. Some studies explore CNNs only for texture feature extraction and use a dedicated classifier apart from the model architecture. Cimpoi et al. [15] was one of the first works on the subject, where the authors compare the efficiency of two different CNN architectures for feature extraction: FC-CNN, which uses a fully connected (FC) layer, and FV-CNN, which uses a Fisher vector (FV) [7] as a pooling method. They demonstrated that, in general, FC features are not that efficient because their output is highly correlated with the spatial order of the pixels. Later on, Condori and Bruno [16] developed a model, called RankGP-3M-CNN, which performs multi-layer feature aggregation employing Global Average Pooling (GAP) to extract the feature vectors of activation maps at different depths of three combined CNNs (VGG-19, Inception-V3, and ResNet50). They propose a ranking technique to select the best activation maps given a training dataset, achieving promising results in some cases but at the cost of increased computational load, since three backbones are needed. Lyra et al. [17] also proposes feature aggregation from multiple convolutional layers, but pooling is performed using an FV-based approach (Multilayer-FV).

Numerous studies propose end-to-end architectures that enable fine-tuning of the backbone for texture recognition. Zhang et al. [18] proposed an orderless encoding layer on top of a DCNN, called Deep Texture Encoding Network (Deep-TEN), which allows images of arbitrary size. Xue et al. [19] introduces a Deep Encoding Pooling Network (DEPNet), which combines features from the texture encoding layer from Deep-TEN and a global average pooling (GAP) to explore both the local appearance and global context of the images. These features are further processed by a bilinear pooling layer [20]. In another work, Xue et al. [21] also combined features from differential images with the features of DEPNet into a new architecture. Using a different approach, Zhai et al. [22] proposed the Multiple-Attribute-Perceived Network (MAP-Net), which incorporated visual texture attributes in a multi-branch architecture that aggregates features of different layers. Later on [9], they explored the spatial dependency among texture primitives for capturing structural information of the images by using a model called Deep Structure-Revealed Network (DSRNet). Chen et al. [23] introduced the Cross-Layer Aggregation of a Statistical Self-similarity Network (CLASSNet). This CNN feature aggregation module uses a differential box-counting pooling layer that characterizes the statistical self-similarity of texture images. More recently, Yang et al. [8] proposed DFAEN (Double-order Knowledge Fusion and Attentional Encoding Network), which takes advantage of attention mechanisms to aggregate first- and second-order information for encoding texture features. Fine-tuning is employed in these methods to adapt the backbone to the new architecture along with the new classification head.

As an alternative to CNNs, Vision Transformers (ViTs) [24] are emerging in the visual recognition literature. Some works have briefly explored their potential for texture analysis through the Describable Textures Dataset (DTD) achieving state-of-the-art results. Firstly, ViTs achieve competitive results compared to CNNs, but the lack of the typical convolutional inductive bias usually results in the need for more training data. To overcome this issue, a promising alternative is to use attention mechanisms to learn directly from text descriptions about images, e.g.using Contrastive Language Image Pre-training (CLIP) [25]. There have also been proposed bigger datasets for the pre-training of ViTs, such as Bamboo [26], showing that these models scale well. Another approach is to optimize the construction of multitask large-scale ViTs such as proposed by Gesmundo [27] with the μ2Net+ method.

2.2. Randomized neural networks for texture analysis

A Randomized Neural Network [10–12], in its simplest form, is a single-hidden-layer feed-forward neural network whose input weights are random, while the weights of the output layer are learned by a closed-form solution, in contrast to gradient-descent-based learning. Recently, some works have investigated RNNs to learn texture features for image analysis. S Junior et al. [28] used small local regions of one image as inputs to an RNN, and the central pixel of the region as the target. The trained weights of the output layer for each image are then used as a texture representation. Ribas et al. [29] improved the previous approach with the incorporation of graph theory to model the texture image.

The training of 1-layer RNNs as employed in previous works is a least-squares solution at the output layer. First, consider $X \in \mathbb{R}^{n \times z}$ as the input matrix with n training samples and z features, and $g = \phi(XW)$ as the forward pass of the hidden layer with a sigmoid nonlinearity, where $W \in \mathbb{R}^{z \times q}$ represents the random input weights for q neurons. Given the desired output labels Y, the output weights f are obtained as the least-squares solution of a system of linear equations:

$$f = Yg^T(gg^T)^{-1}, \qquad (1)$$

where $g^T(gg^T)^{-1}$ is the Moore–Penrose pseudo-inverse [30,31] of matrix g.

[1] https://github.com/scabini/RADAM

An important aspect of RNNs is the generation of random weights for the first layer. Evidence suggests that this choice has little impact once the weights are fixed. In this sense, a common trend among previous works is the use of the Linear Congruential Generator (LCG), a simple pseudo-random number generator in the form of $x_{k+1} = (ax_k + b) \mathrm{mod} c$.

The RNN can be used as a randomized autoencoder (RAE) [13] by considering the input feature matrix X as the target output $Y = X$. In this sense, the model is composed of a random encoder and a least-squares-based decoder that can map the input data. Kasun et al. [13] also suggests the use of random orthogonal weights [32] for the initialization of the encoder. In this way, the weight matrix f will represent the transformation of the projected random space back into the input data X (output).

3. RADAM for texture feature aggregation and encoding

The majority of the previous works on deep texture recognition consider end-to-end models that couple new modules around a pre-trained backbone and involve fine-tuning the new architecture on specific texture recognition tasks. Here, we propose a new feature encoding module that acts on the backbone only as a feature extractor, and a dedicated classifier is applied over the obtained features (no backbone fine-tuning is done). The main idea of the proposed RADAM method is to use multi-depth feature aggregation and randomized pixel-wise encoding to compose a single feature vector, given an input image processed by the backbone. First of all, consider an input image $I \in \mathbb{R}^{w_0 \times h_0 \times 3}$ fed into a backbone $B = (d_1, \ldots, d_n)$, consisting of n blocks of convolutional layers. An activation map, i.e., the output of any convolutional block given I, is a 3-dimensional tensor (ignoring the batch dimension, for simplicity) $X_i \in \mathbb{R}^{w_i \times h_i \times z_i}$. The process of feature aggregation consists of combining the outputs of different activation maps at different depths. To that end, we divide the backbone into a fixed number of blocks according to different depths. This division is made to keep a fixed number of blocks for feature extraction, regardless of the total depth of the backbone architecture.

3.1. Pre-trained deep convolutional networks: backbone selection

Most previous works on texture analysis consider pre-trained ResNets [33] (18 or 50) as backbones. Here, we consider the output of five blocks of layers according to the ResNet architecture, meaning that five activation maps are considered for feature aggregation. Additionally, we consider the ConvNeXt architecture [34], a more recent method with promising results in image recognition. For this backbone, we consider the activation maps from the four blocks of layers according to the architecture described in the original work. More specifically, the following ConvNeXt configurations are used, with their corresponding number of channels (z_i) of each block:

- ConvNeXt-nano[2]: $z_i = (80, 160, 320, 640)$;
- ConvNeXt-T: $z_i = (96, 192, 384, 768)$;
- ConvNeXt-B: $z_i = (128, 256, 512, 1024)$;
- ConvNeXt-L: $z_i = (192, 384, 768, 1536)$;
- ConvNeXt-XL: $z_i = (256, 512, 1024, 2048)$.

[2] This variant was not presented in the original paper but is available in the PyTorch Image Models library [35] (version 0.6.7) https://github.com/huggingface/pytorch-image-models/

3.2. Deep activation map preparation

Given each deep activation map X_i, we apply a depth-wise l_p-normalization ($p = 2$, i.e., Euclidean norm):

$$X_i(:,:,j) = \frac{X_i(:,:,j)}{\max(||X_i(:,:,j)||_2)}, \tag{2}$$

where $X_i(:,:,j)$ represents the 2-dimensional activation map at each channel $j \in z_i$ with spatial sizes (w_i, h_i).

For feature aggregation, we propose to concatenate the activation maps along the third dimension (z_i). However, each map X_i initially has a different spatial dimension w_i and h_i. To overcome this, we simply resize all activation maps with bilinear interpolation using the spatial dimensions of $X_{\frac{n}{2}}$, $(w_{\frac{n}{2}}, h_{\frac{n}{2}})$, as the target sizes. In other words, we consider the spatial dimensions at the middle of the backbone as our anchor size, meaning that some activation maps will require upscaling (if $i > \frac{n}{2}$) and others downscaling (if $i < \frac{n}{2}$). Naturally, the information from activation maps at higher depths receives higher priority considering that upscaling preserves more information than downscaling. These assumptions consider the most common structure of convolutional architectures where the spatial size decreases with layer depth. Nonetheless, the idea is to keep all activation maps with a fixed spatial dimension. From now on, we will refer to spatial dimensions of all X_i as $w = w_{\frac{n}{2}}$ and $h = h_{\frac{n}{2}}$. For an input size of 224×224, this results in $w = h = 28$ for the backbones explored in this work. The concatenation of activation maps is then performed as

$$X' = [X_1; \ldots; X_n] \in \mathbb{R}^{w \times h \times z} \rightarrow X' \in \mathbb{R}^{wh \times z}, \tag{3}$$

where $[.;.]$ denotes the concatenation along the third dimension, and $z = \sum_i z_i$ is the resulting number of channels after concatenation. Considering common convolutional architectures where $z_i < z_{i+1}$, activation maps from higher depths have a higher influence on the overall z features. Additionally, the 2-dimensional activation map at each channel z_i is flattened, resulting in the reshaped 2D representation X' with sizes wh-by-z, which we refer to as an aggregated activation map. These steps are illustrated in Fig. 1, which reports the overall structure of the proposed method.

3.3. Pixel-wise randomized encoding

The aggregated activation map of a single image is used to train an RAE considering each spatial point, or pixel (row of X'), as a sample and each channel (column of X') as a feature. In this sense, the method also works with arbitrary input sizes (if accepted by the backbone) since the spatial dimensions only affect the number of training samples for the RAE. Intuitively, larger input sizes may improve the RAE training (see Fig. 3(b)), but also increases the backbone cost significantly. Moreover, considering that the spatial organization of the pixels is lost due to the flattening procedure of X', we add a positional encoding composed of sine and cosine functions of different frequencies with dimension z extended for 2 spatial dimensions as in Wang and Liu [36]:

$$\begin{aligned}
\mathrm{PE}(x, y, 2i) &= \sin\left(\frac{x}{10{,}000^{4i/z}}\right),\\
\mathrm{PE}(x, y, 2i+1) &= \cos\left(\frac{x}{10{,}000^{4i/z}}\right),\\
\mathrm{PE}(x, y, 2j + z/2) &= \sin\left(\frac{y}{10{,}000^{4j/z}}\right),\\
\mathrm{PE}(x, y, 2j + 1 + z/2) &= \cos\left(\frac{y}{10{,}000^{4j/z}}\right),
\end{aligned}$$

where $x \in w$ and $y \in h$, which is then added to the aggregated activation map via element-wise sum:

$$X' = X' \oplus PE. \tag{4}$$

The use of positional encoding is considered to evaluate whether or not adding spatial information to the aggregated deep features could improve texture discrimination. In the context of

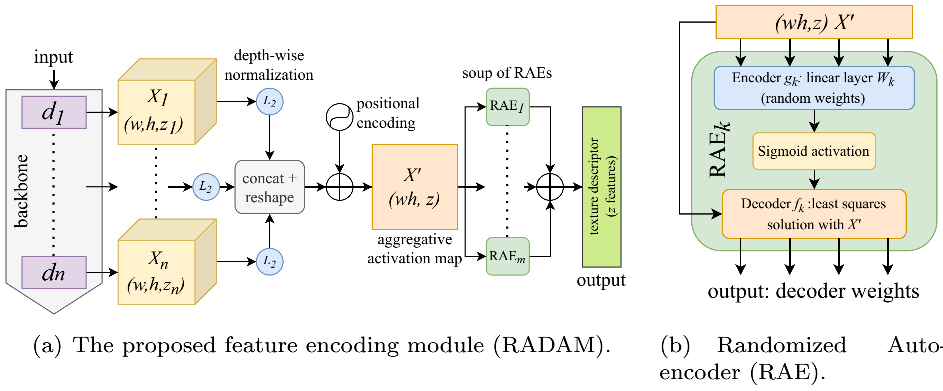

input
backbone
d_1
dn
X_1 (w,h,z_1)
X_n (w,h,z_n)
depth-wise normalization
L_2
concat + reshape
positional encoding
X' (wh, z)
aggregative activation map
soup of RAEs
RAE$_1$
RAE$_m$
texture descriptor (z features)
output
(wh,z) X'
Encoder g_k: linear layer W_k (random weights)
Sigmoid activation
Decoder f_k :least squares solution with X'
RAE$_k$
output: decoder weights
(a) The proposed feature encoding module (RADAM).
(b) Randomized Auto-encoder (RAE).

Fig. 1. Illustration of the proposed RADAM architecture (a), given an input image to a final descriptor. The RAE is shown in detail (b), which is a simple 1-layer auto-encoder solved through least-squares, where we use the decoder weights as descriptors (summed for m RAEs).

our method, we employ RAEs over these features to encode the entire information in an orderless fashion since the RAE is trained using each z-dimensional deep feature as a training sample. According to the training algorithm of the RAE, the order of the samples does not matter, which differs from a typical SGD, for instance. Therefore, this process differs from both global pooling and the flattening of the features. The positional encoding is simply adding spatial information into the deep features about how they were organized before in the 3D aggregated activation map, which is considered by the RAE when learning their representations. In Section 4, we evaluate the impact of (not) using positional encoding.

After summing the positional encoding to X', the first step of the RAE is to project the inputs using a random fully-connected layer with weights $W_k \in \mathbb{R}^{z\times q}$, followed by a sigmoid nonlinearity. The weights are generated using the LCG for simplicity and better replicability, followed by standardization (zero centered, unity variance) and orthogonalization [32]. These configurations were chosen according to previous works [13,29]. As for the LCG parameters, we use $a = 75$, $b = 74$, and $c = 2^{16} + 1$, starting with $x = 0$, which is a classical configuration according to the ZX81 computer from 1981. Here k works like a seed for random sampling, denoting a starting index inside the LCG space generated with the given configuration. More details on LCG weights are given in the Supplementary Material, such as an ablation on the impacts of different LCG configurations. The forward pass of the encoder $g_k \in \mathbb{R}^{wh\times q}$ for all samples is then obtained as

$$g_k = \phi(X'W_k), \tag{5}$$

and the decoder weights $f_k \in \mathbb{R}^{z\times q}$ are obtained as the least-squares solution described in Eq. (1), changing the target Y to X':

$$f_k = X' g_k^T (g_k g_k^T)^{-1}. \tag{6}$$

The main idea of employing an individual randomized neural network for each image is to use the output weights themselves as a representation. In the case of RAEs, the output layer has the same dimension as the input layer. Therefore, a single hidden neuron ($q = 1$) is considered to maintain the dimensionality. In this sense, the resulting decoder weights are represented by

$$f_k = (\nu_1, \dots, \nu_z), \tag{7}$$

where ν_i represents the connection weight between the single hidden neuron and the output i, corresponding to feature $i \in z$.

A single-neuron RAE may be limited in encoding enough information contained in the deep activation maps. Therefore, we propose an ensemble of models or, as recently introduced [37], a model "soup", which is achieved by combining the weights of m parallel models. Here, each model is an RAE with a different random encoder (using a different LCG seed), and the combination is performed by summing the decoder weights

$$\varphi_m = \Big(\sum_{k=1}^{m} f_k(\nu_1), \dots, \sum_{k=1}^{m} f_k(\nu_z)\Big). \tag{8}$$

It is important to note that the encoders g_k of each of the m RAEs have a different random weight initialization. This is achieved by creating an LCG sequence of size mz so that we have z weights for each of the m RAEs. The structure of the RAE is illustrated in Fig. 1, and following the whole RADAM pipeline shown in Fig. 1, a texture representation, or feature vector φ_m, is obtained for the input image. The code for all these steps is available in the Supplementary Material and in our online repository[1]. The feature vectors φ_m are then used to train a linear classifier for a given texture recognition task (more details on the classifier can be consulted in Section 4.2.1).

4. Experiments and results

4.1. Setup

Our model is implemented using PyTorch [38] (except for the classification step), making it easier to couple RADAM with several methods implemented in this library. The classification step is performed using Scikit-learn [39] (with standard hyperparameters and no tuning). We measure our results by the average classification accuracy and corresponding standard deviation, when applicable (depending on the dataset). For the backbones, we consider the PyTorch Image Models library [35] (version 0.6.7), which contains several pre-trained computer vision methods. In the Supplementary Material, we present the main code for RADAM, and the complete implementation including scripts for experimentation can be consulted in our GitHub repository.[1]

Seven texture datasets are used for evaluation purposes in this paper, of which the following two variants of the Outex dataset [40] were used for analyzing the RADAM method alone:

- *Outex10*: Composed of 4320 grayscale images in 24 different texture classes. The training split contains 480 images and the test split contains 3840 images. This dataset focuses on rotation invariance, so the images are rotated at 9 different angles;
- *Outex13*: This Outex suite holds 1360 RGB images divided into 68 texture classes, and evaluates color texture recognition. It contains 680 images in the training split and 680 images in the test split.

The following five datasets are used for comparisons with other methods (some samples are shown in Fig. 2):

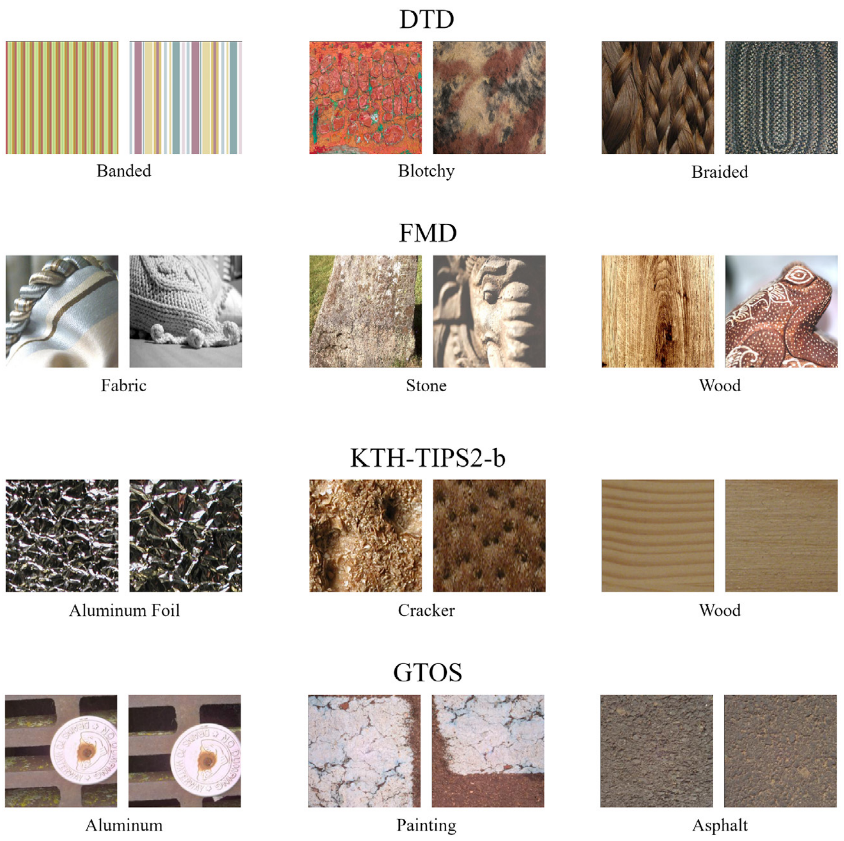


Fig. 2. Some samples from four datasets that are employed in this work for comparison with existing literature methods. We show two samples from each of three classes, for each dataset. GTOS-Mobile, though not depicted, shares substantial similarities with the GTOS dataset.

- *Describable Texture Dataset (DTD)* [7]: Composed of 5640 images in 47 different texture classes, evaluated by the 10 provided splits for training, validation, and test;
- *Flickr Material Dataset (FMD)* [41]: Holds 1000 images representing 10 material categories, and validation is done through 10 repetitions of 10-fold cross-validation;
- *KTH-TIPS2-b* [42]: Contains 4752 images of 11 different materials. This dataset has a fixed set of 4 splits for 4-fold cross-validation;
- *Ground Terrain in Outdoor Scenes (GTOS)* [21]: This dataset represents 34,105 images divided into 40 outdoor ground materials classes. There is also a fixed set of 5 train/test splits;
- *GTOS-M (Mobile)* [19]: Consists of 100,011 images captured from a mobile phone of 31 different outdoor ground materials, and contains a single train/test split.

4.2. Analysis of RADAM properties

Our first experimental evaluation concerns aspects of the proposed RADAM method. In the Supplementary Material, we show an additional analysis of the impacts caused by different random weights (LCG configurations) and concluded that they are minimal, corroborating previous works. In the following, we evaluate and discuss other aspects of RADAM.

4.2.1. Positional encoding and different classifiers

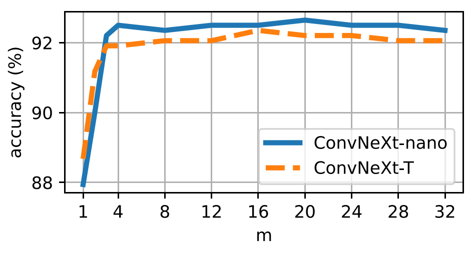
We evaluate two design choices for the RADAM pipeline, the use of positional encoding and the classifier. The method is compared with or without encoding under four different classifiers:

- Linear Discriminant Analysis (LDA) [43]: using a least-squares solution with automatic shrinkage using the Ledoit–Wolf lemma;
- Support Vector Machines (SVM) [44]: using a linear kernel with $C = 1$;
- Random Forests (RF) [45]: using 100 trees, the Gini impurity, and the square of the number of features when looking for the best splits;
- Multilayer Perceptron (MLP): composed of a hidden layer with 100 neurons, a softmax output layer, ReLU activations, and trained using Adam with β equal to 0.9 or 0.999, a fixed learning rate of 10^{-3}, and 200 epochs.

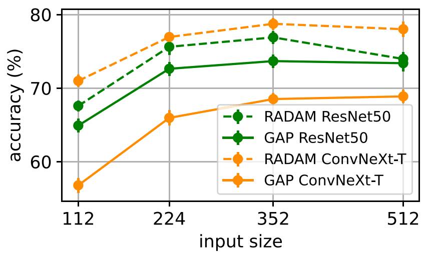
Table 1
Ablations with positional encoding and different classifiers, using RADAM ($m = 1$) with different backbones. Results are measured by classification accuracy considering the single train/test split of Outex10, or the average and standard deviation for 10 repetitions with the RF and MLP classifiers.

Encoding	Backbone	LDA	SVM	RF	MLP
none	ResNet18	77.5	83.4	85.5±1.5	81.9±2.0
positional	ResNet18	79.5	84.7	87.3±0.5	82.7±2.1
none	ConvNeXt-nano	82.4	89.6	81.2±1.3	85.7±2.3
positional	ConvNeXt-nano	85.9	89.6	82.4±1.0	84.7±2.2
none	ResNet50	86.0	87.4	90.6±0.9	86.0±0.9
positional	ResNet50	87.4	87.4	87.8±0.9	84.5±1.3
none	ConvNeXt-T	87.8	91.1	82.6±1.1	86.7±1.4
positional	ConvNeXt-T	89.4	91.1	87.0±1.2	86.6±1.3

Since the evaluation of positional encoding concerns the spatial properties of texture, we consider the Outex10 benchmark, which focuses on rotation invariance. As the results in Table 1 demonstrate, positional encoding improves or maintains performance in general. On the other hand, SVM provides the best results in most cases while gaining less improvement from positional encoding.



(a) Soup size ablation.



(b) Input size ablation.

Fig. 3. Impacts of changing the number of RAEs (m) in the proposed method, considering different backbones on the Outex13 dataset (a), and (b) the image input size on the DTD dataset (with $m = 4$). All other experiments in this paper consider $m = 4$ and input size 224×224.

Nevertheless, we keep the positional encoding in our architecture since the additional cost is negligible compared to the potential gains. The SVM is also used as the classifier for all the following experiments.

4.2.2. Soup size

The only free parameter of the proposed RADAM method is the number of RAEs to be combined, i.e., m. We evaluated m ranging from 1 to 32 and the results are shown in Fig. 3(a) for different backbones in the Outex13 dataset. We observe significant gains for m from 1 to 4, while for larger values performance tends to stabilize. These results indicate that $4 \leq m \leq 8$ is a good approach for a balance between performance and cost since no significant gains are achieved above that. All the following experiments in this paper are performed using $m = 4$.

The effects observed when increasing m are expected considering what is usually seen in model ensembles, or "model soups" [37], where the combination of models trained separately may be beneficial. On the other hand, our encoders are random, and each one has different weights. However, even if each encoder creates a different random projection of the input, the decoders learn to transform the projection back to the same feature space. In other words, the RAEs learn different encoding-decoding functions for the same input that, when combined, provide a better representation in our feature extraction use case.

4.3. Comparison with other pooling techniques

To show the gains of RADAM over other pooling approaches, we performed additional experiments using Global Average Pooling (GAP), Global Max Pooling (GMP), and Global L-p Pooling (GLpP, a.k.a. power-average pooling, and we use $p = 2$). These techniques are applied over pre-trained backbones to obtain a feature vector, given input images, which are then used to train an SVM using the same configurations considered for RADAM. The obtained results are shown in Table 2 for four backbones and two challenging texture recognition datasets (DTD and FMD). We indicate the pre-training dataset used for the backbone using "in", e.g.in IN-21K (ImageNet-21K [46]), and ImageNet-1K was used when not stated. In summary, GAP is superior to the other pooling techniques in all cases. On the other hand, RADAM achieves considerably superior results compared to GAP, with gains above 10% in absolute classification accuracy in some cases. We also evaluate the impact of changing the input sizes when using RADAM or GAP with ResNet50 or ConvNeXt-T, and the results are shown in Fig. 3(b) for DTD. We observe that increasing the input size usually improves performance, except for RADAM with ResNet50 with input size 512. Nevertheless, we keep the input size for the following experiments at 224 to maintain the fairness of comparison with other methods from the literature. These results corroborate the effectiveness and scaling potential of RADAM for texture feature extraction compared to other pooling techniques, especially when considering ConvNeXt backbones.

Table 2
Classification accuracy (linear SVM) when using image features obtained from the backbones with different pooling techniques and the proposed RADAM. GAP agg. concatenates features from each of the n feature blocks of the backbones (the same way we approach them with RADAM).

Method	Backbone	DTD	FMD
GMP	ResNet18	62.3±0.8	75.6±0.5
GLpP	ResNet18	63.7±1.1	77.0±0.6
GAP	ResNet18	64.1±1.1	77.1±0.7
GAP agg.	ResNet18	64.6±1.2	77.2±0.6
RADAM	ResNet18	68.1±1.0	77.7±0.5
GMP	ConvNeXt-nano	51.7±0.9	62.4±0.8
GLpP	ConvNeXt-nano	44.0±0.5	50.4±0.8
GAP	ConvNeXt-nano	61.9±1.0	73.8±0.9
GAP agg.	ConvNeXt-nano	71.7±0.6	84.4±0.4
RADAM	ConvNeXt-nano	74.9±0.7	87.1±0.4
GMP	ConvNeXt-B	59.5±1.2	77.0±0.8
GLpP	ConvNeXt-B	59.1±1.3	70.6±0.8
GAP	ConvNeXt-B	69.2±1.3	83.6±0.5
GAP agg.	ConvNeXt-B	73.5±1.0	86.7±0.5
RADAM	ConvNeXt-B	76.4±0.9	90.2±0.2
GMP	ConvNeXt-B in IN-21K	70.6±0.8	84.2±0.4
GLpP	ConvNeXt-B in IN-21K	69.7±0.9	72.0±0.6
GAP	ConvNeXt-B in IN-21K	79.2±0.6	91.9±0.4
GAP agg.	ConvNeXt-B in IN-21K	80.4±1.0	92.3±0.3
RADAM	ConvNeXt-B in IN-21K	82.8±0.9	94.0±0.2
GAP	ConvNeXt-T	66.0±1.0	79.0±0.4
GAP agg.	ConvNeXt-T	73.7±0.9	83.6±0.5
RADAM	ConvNeXt-T	77.0±0.7	88.7±0.4
GAP	ConvNeXt-T in IN-21K	78.4±0.7	91.4±0.3
GAP agg.	ConvNeXt-T in IN-21K	77.3±0.9	89.9±0.5
RADAM	ConvNeXt-T in IN-21K	81.4±0.7	93.0±0.3
GAP	ConvNeXt-L	70.9±0.9	84.6±0.4
GAP agg.	ConvNeXt-L	73.4±0.6	86.4±0.5
RADAM	ConvNeXt-L	77.4±1.1	89.3±0.3
GAP	ConvNeXt-L in IN-21K	80.4±0.9	92.2±0.4
GAP agg.	ConvNeXt-L in IN-21K	81.9±0.6	93.4±0.4
RADAM	ConvNeXt-L in IN-21K	84.0±1.0	95.2±0.4
GAP	ConvNeXt-XL in IN-21K	81.3±1.0	92.4±0.3
GAP agg.	ConvNeXt-XL in IN-21K	82.0±1.1	93.9±0.4
RADAM	ConvNeXt-XL in IN-21K	83.7±0.9	95.2±0.3

Additionally, we evaluate GAP agg., which aggregates the GAP from each of the n feature blocks and returns an image representation with z features (as RADAM). The results are also shown in Table 2, where it is possible to observe that RADAM surpasses GAP agg. by a considerable margin, in all backbones and datasets. GAP agg. usually improves over the standard GAP, which is to be expected since more features are being added. However, the standard GAP sometimes performs better than this simple aggregation by

Table 3
Classification accuracy of different methods on texture benchmarks. The used backbones are separated into row blocks according to their computational budget, the input size is indicated in parentheses (224 × 224 was used when not stated), and the two best results in each block are highlighted in **bold type**. Methods indicated by a † symbol mean that we are citing the results from their papers. Results in blue show the previously published state-of-the-art on each dataset (some are given in Table 4), and red represents our results matching or above that.

method	backbone	DTD	FMD	KTH-2-b	GTOS	GTOS-M
GAP	ResNet18	64.1$_{\pm1.1}$	77.1$_{\pm0.7}$	83.0$_{\pm2.9}$	78.8$_{\pm1.7}$	75.1
DeepTEN †	ResNet18 (352)					76.1
DEPNet †	ResNet18					82.2
MAPNet †	ResNet18	69.5$_{\pm0.8}$	80.8$_{\pm1.0}$	80.9$_{\pm1.8}$	80.3$_{\pm2.6}$	83.0
DSRNet †	ResNet18	71.2$_{\pm0.7}$	81.3$_{\pm0.8}$	81.8$_{\pm1.6}$	81.0$_{\pm2.1}$	**83.7**
CLASSNet †	ResNet18	**71.5**$_{\pm0.4}$	**82.5**$_{\pm0.7}$	**85.4**$_{\pm1.1}$	**84.3**$_{\pm2.2}$	**85.3**
RADAM (ours)	ResNet18	68.1$_{\pm1.0}$	77.7$_{\pm0.5}$	84.7$_{\pm3.6}$	80.6$_{\pm1.7}$	79.5
GAP	ConvNeXt-nano	61.9$_{\pm1.0}$	73.8$_{\pm0.9}$	82.3$_{\pm3.0}$	78.9$_{\pm1.8}$	74.0
RADAM (ours)	ConvNeXt-nano	**74.9**$_{\pm0.7}$	**87.1**$_{\pm0.4}$	**89.6**$_{\pm3.8}$	**83.7**$_{\pm1.5}$	81.8
GAP	ResNet50	72.6$_{\pm0.9}$	84.5$_{\pm0.6}$	85.4$_{\pm2.5}$	79.5$_{\pm1.7}$	76.1
DeepTEN †	ResNet50 (352)	69.6	80.2$_{\pm0.9}$	82.0$_{\pm3.3}$	84.5$_{\pm2.9}$	
MAPNet †	ResNet50	76.1$_{\pm0.6}$	85.2$_{\pm0.7}$	84.5$_{\pm1.3}$	84.7$_{\pm2.2}$	86.6
DSRNet †	ResNet50	**77.6**$_{\pm0.6}$	86.0$_{\pm0.8}$	85.9$_{\pm1.3}$	**85.3**$_{\pm2.0}$	**87.0**
CLASSNet †	ResNet50	74.0$_{\pm0.5}$	86.2$_{\pm0.9}$	87.7$_{\pm1.3}$	**85.6**$_{\pm2.2}$	85.7
DFAEN †	ResNet50	73.2	**86.9**	86.3		**86.9**
RADAM (ours)	ResNet50	75.6$_{\pm1.1}$	85.3$_{\pm0.4}$	**88.5**$_{\pm3.2}$	81.8$_{\pm1.1}$	81.0
GAP	ConvNeXt-T	66.0$_{\pm1.0}$	79.0$_{\pm0.4}$	88.1$_{\pm3.8}$	80.9$_{\pm1.6}$	78.2
RADAM (ours)	ConvNeXt-T	**77.0**$_{\pm0.7}$	**88.7**$_{\pm0.4}$	**90.7**$_{\pm4.0}$	84.2$_{\pm1.7}$	85.3

concatenation. These results suggest that better aggregation techniques are needed for improving texture feature extraction. In this sense, the proposed RADAM shows again interesting gains, resulting in state-of-the-art performance on all benchmarks we considered, as we discuss in the following section.

4.4. Comparison with the state-of-the-art

Finally, we compare RADAM with several state-of-the-art methods on five challenging texture recognition datasets. Table 3 shows the results of methods using the ResNet18 and 50 and ConvNeXt-nano and T backbones. The table is organized into separate rows according to the computational budget of the backbones. The first comparison section contains methods using ResNet18 and ConvNeXt-nano since they have a similar computational budget. In this scenario, RADAM achieves competitive performance on KTH when using ResNet18 but is less effective on other datasets. Nevertheless, it is worth noting that RADAM with ResNet18 surpasses GAP in all cases and does not require fine-tuning, offering a much cheaper and simple alternative to other methods that require fine-tuning. Using ConvNeXt-nano, RADAM significantly outperforms GAP and achieves the best results on all datasets except GTOS and GTOS-M, and at a similar inference budget to using ResNet18.

Considering the results within the ResNet50 budget, RADAM performs much better compared to ResNet18. Competitive results are achieved on DTD and FMD, and also the best results on KTH. We observe again that on GTOS, although surpassing GAP, RADAM performs below the compared methods that perform fine-tuning. These results are expected since both GTOS datasets are by far the biggest datasets among the ones considered, which allows better fine-tuning of the millions of parameters of the backbones. However, this procedure means that a considerable computational load is necessary to train these methods relying on fine-tuning. On the other hand, by using ConvNeXt-T, RADAM surpasses GAP by a significant margin and offers a competitive performance on GTOS compared to the fine-tuning-based methods. It also achieves the second-best result on DTD and the best results on FMD and KTH. The performance on FMD also surpasses the previous state-of-the-art considering all results available in the literature (which considers more complex backbones, see Table 4). In general, we notice that in most cases, the GAP of ConvNeXt performed below the GAP of their corresponding ResNets (in terms of cost according to the blocks of the table). In contrast, RADAM achieves considerably better results when coupled with ConvNeXts. Therefore, the results clearly emphasize the gains of RADAM over the potential difference between different backbones.

Table 4 concerns methods with an increased cost or improved pre-training, in contrast to those compared before. Here we include RADAM using ConvNeXt-T, B, L, and XL, with ImageNet-1k or 21k pre-training, against several works including very recent methods, such as ViTs. In the first block, RADAM outperforms GAP in all cases, and also the other compared methods on FMD, KTH, and GTOS. On DTD and GTOS-M, RADAM achieves the second-best result when using ConvNeXt-L. The results obtained on FMD using both ConvNeXt-B and L backbones surpasses the previously published state-of-the-art on this dataset.

The second block highlights the benefits of employing better pre-training. For instance, ConvNeXts T, B, L, and XL surpass the previously published state-of-the-art on FMD by using GAP. ConvNeXt-XL also achieves this for KTH and GTOS-M. Nevertheless, this is not enough to surpass previous methods on DTD or GTOS. On the other hand, RADAM is able to overcome the previously published state-of-the-art in all benchmarks. It consistently outperforms GAP, and also all other methods across every benchmark when using the ConvNeXt-XL backbone, establishing a new state-of-the-art. When using ConvNeXt-B, it also surpasses the previous state-of-the-art in all cases except for DTD. Using ConvNeXt-T, it overcomes the state-of-the-art on FMD and offers competitive performance on the other benchmarks while having a lower inference cost, considering the size of this backbone. With ConvNeXt-L, RADAM also establishes a new state-of-the-art on DTD.

In conclusion, these results underline the effectiveness of RADAM as a robust method for texture feature extraction followed by a linear SVM classification. Its performance, particularly when using ConvNeXt backbones, offers high-level texture discrimination without the need for fine-tuning, which saves considerable resources in training time, as we better discuss in the following.

Table 4
Classification accuracy of methods employing more complex backbones and pre-trainings. The first block represents methods with ImageNet1k pre-training only, and the second block includes different pre-trainings with bigger datasets. Results in show the previously published state-of-the-art on each dataset (some are given in Table 3), represents the results matching or above that, and the two best results in each block are highlighted in **bold type**.

method	backbone	DTD	FMD	KTH-2-b	GTOS	GTOS-M
GAP	ConvNeXt-B	$69.2_{\pm1.3}$	$83.6_{\pm0.5}$	$86.2_{\pm6.8}$	$80.4_{\pm1.9}$	80.1
GAP	ConvNeXt-L	$70.9_{\pm0.9}$	$84.5_{\pm0.4}$	$88.0_{\pm4.0}$	$81.3_{\pm2.0}$	85.5
GAP	Densenet161	$71.8_{\pm1.4}$	$85.2_{\pm0.7}$	$87.4_{\pm2.1}$	$80.4_{\pm1.9}$	81.5
DFAEN †	Densenet161	76.1	87.6	86.6		**86.9**
Multilayer-FV †	EfficientNet-B5 (512)	**78.9**	88.7	82.9		
RankGP-3M-CNN++ †	(3 backbones)		$86.2_{\pm1.4}$	**91.1**$_{\pm4.5}$		
RADAM (ours)	ConvNeXt-B	$76.4_{\pm0.9}$	**90.2**$_{\pm0.2}$	$87.7_{\pm5.6}$	**84.1**$_{\pm1.6}$	82.2
RADAM (ours)	ConvNeXt-L	**77.4**$_{\pm1.1}$	**89.3**$_{\pm0.3}$	**89.3**$_{\pm3.4}$	**84.0**$_{\pm1.8}$	**85.8**
GAP	ConvNeXt-T in IN-21K	$78.4_{\pm0.7}$	$91.4_{\pm0.3}$	$89.6_{\pm4.4}$	$82.8_{\pm2.6}$	84.0
GAP	ConvNeXt-B in IN-21K	$79.2_{\pm0.6}$	$91.9_{\pm0.4}$	$90.4_{\pm4.5}$	$84.4_{\pm1.8}$	86.5
GAP	ConvNeXt-L in IN-21K	$80.4_{\pm0.9}$	$92.2_{\pm0.4}$	$88.7_{\pm3.7}$	$83.2_{\pm1.4}$	84.5
GAP	ConvNeXt-XL in IN-21K	$81.3_{\pm1.0}$	$92.4_{\pm0.3}$	**93.7**$_{\pm4.0}$	$84.5_{\pm2.1}$	**89.8**
fine-tuning only †	ViT-B/16 in Bamboo 69m	81.2				
CLIP zero-shot †	ViT-L/14 (336) in WIT 400m	83.0				
μ2Net+ †	ViT-L/16 (384) in IN-21K	82.2				
RADAM (ours)	ConvNeXt-T in IN-21K	$81.4_{\pm0.7}$	$93.0_{\pm0.3}$	$91.0_{\pm4.9}$	$85.4_{\pm1.6}$	86.5
RADAM (ours)	ConvNeXt-B in IN-21K	$82.8_{\pm0.9}$	$94.0_{\pm0.2}$	$91.8_{\pm4.1}$	**86.6**$_{\pm1.7}$	87.1
RADAM (ours)	ConvNeXt-L in IN-21K	**84.0**$_{\pm1.0}$	**95.2**$_{\pm0.4}$	$91.3_{\pm4.1}$	$85.9_{\pm1.6}$	87.3
RADAM (ours)	ConvNeXt-XL in IN-21K	**83.7**$_{\pm0.9}$	**95.2**$_{\pm0.3}$	**94.4**$_{\pm3.8}$	**87.2**$_{\pm1.9}$	**90.2**

4.5. Challenging texture instances

This section discusses the cases with the highest error rates for the two most challenging datasets (DTD and GTOS). We compare the performance of RADAM and GAP in ConvNeXt backbones, showing the confusion matrix (classes are sorted alphabetically), and the precision and recall metrics (averaged for all classes). The results for the DTD dataset are shown in Fig. 4, along with examples from the most challenging classes. In this case, RADAM achieves the best performance using ConvNeXt-L, while GAP needs the bigger version, ConvNeXt-XL (both pre-trained on IN-21k), to achieve the best classification accuracy. In terms of precision and recall, RADAM also achieves the best results (0.84 against 0.81). Regarding the misclassification, it is possible to notice that the class "blotchy" is the most challenging for both methods. The examples in Fig. 4(c) show how complex this class is, where the instances contain various types of objects and materials. The results indicate that the methods struggle to model the "blotchy" concept, which is more intuitive for humans to understand. GAP with ConvNeXt-XL achieves 50.8% classification accuracy for this class, while RADAM with ConvNeXt-L achieves 54.3%. While this is the class with the lowest accuracy rate, the misclassifications (confusion) are spread across various classes, with the most common one being "smeared". For this class, GAP achieves a 70.8% overall accuracy rate, and RADAM achieves 75.5%. For GAP and RADAM, 10% of the samples are misclassified as smeared, and the percentage of smeared samples that are misclassified as blotchy are 14.3% and 12%, respectively. In general, RADAM improved the classification of these classes and reduced the misclassification rates.

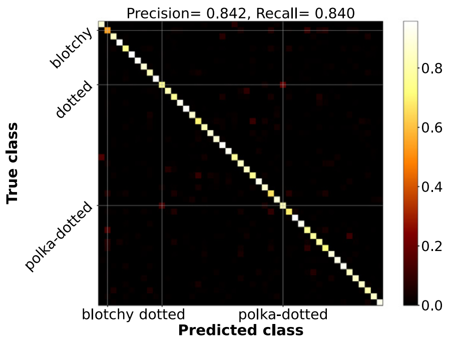
The classes of DTD with the highest confusion rate are "dotted" and "polka-dotted". Their similarity is noticeable, as the examples in Fig. 4(d and e) show, and the models struggle to differentiate the distribution of bubbles with uniform or varied sizes. Nevertheless, RADAM consistently improves the overall accuracy rate of both classes. For the class "dotted", GAP has a 71.3% accuracy rate while RADAM achieves 75.8%. For "polka-dotted", the rates are 73.3% and 82.5% for GAP and RADAM, respectively. While their overall accuracy rate is relatively high, the misclassifications are concentrated between them. GAP has a 20.3% misclassification rate of "dotted" samples as "polka-dotted", and 20.8% for the other way around. In this case, RADAM also improves the performance, showing a reduction to 17.8% misclassification rate of "dotted" as "polka-dotted", and 12.8% for the opposite case.

The results for GTOS are shown in Fig. 5 for GAP and RADAM using ConvNeXt-XL. The most challenging texture classes in this dataset in terms of overall accuracy, for both methods, are "rust cover" and "mud puddle". For the former, GAP achieves a 50.7% classification accuracy, and 34.7% of the samples are misclassified as "aluminum". RADAM shows a slightly inferior performance, with a 48.6% accuracy rate and 41.1% misclassification rate as "aluminum". As Fig. 5(c and d) shows, the images from these classes may contain similar elements such as rusty parts in the aluminum pieces, and the background grids, which proved to be a challenge for the methods. On the other hand, RADAM shows the best precision and recall for this dataset, and also improves the performance for other classes. For the "mud puddle" class, GAP correctly identifies 48% of the samples, while RADAM achieves 52.9%.

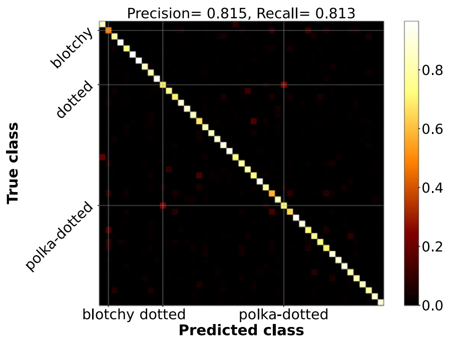
The classes "asphalt stone" and "stone asphalt" from GTOS also present a high confusion rate, as they are particularly similar (see Fig. 5(e and f). GAP correctly predicts 79.2% of the samples from the "asphalt stone" class, while RADAM improves the accuracy to 92.3%. For "stone asphalt", the classification accuracy is 52% and 53.9% for GAP and RADAM, respectively. As for the confusion cases, 46% of the "stone asphalt" samples are misclassified as "asphalt stone" by GAP, while RADAM reduces the error to 45.4%.

4.6. Feature extraction cost versus performance

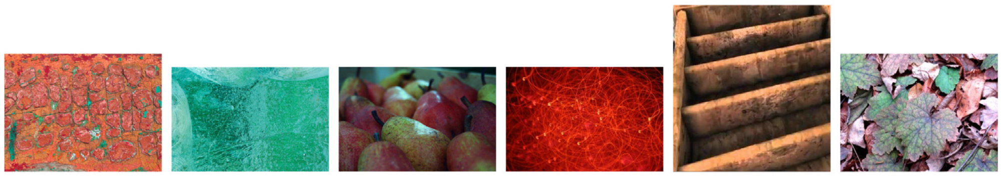
Additional analysis is performed to better understand the balance between the classification performance and computational budget of the compared texture recognition methods. We consider the inference costs in terms of GFLOPs and the number of parameters according to the backbone used by each method since this is the most resource-demanding step of every pipeline. One important aspect here is that the input size greatly impacts the FLOP count of the methods (check input sizes in parentheses in Table 3). Most works consider 224 × 224 inputs (the same input size employed by RADAM), and we assume this same size when not stated by the authors. For this analysis, we also are not considering the preparation of the backbone either in terms of pre-training cost or



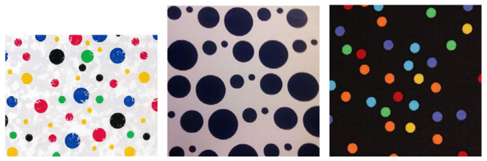
(a) RADAM with ConvNeXt-L.



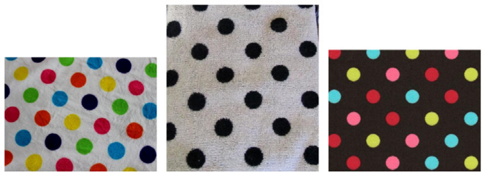
(b) GAP with ConvNeXt-XL.



(c) Samples from the most challenging DTD class: "blotchy".

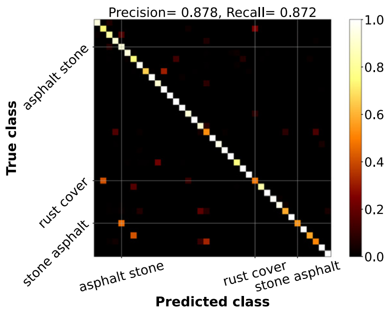


(d) Samples from the class "dotted".

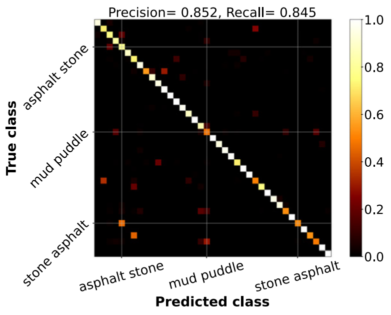


(e) Samples from the class "polka-dotted".

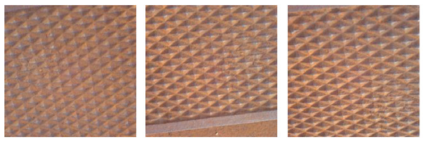
Fig. 4. Confusion matrix and image samples from the most challenging classes of the DTD benchmark. The confusion matrix is normalized by the total number of samples per class after the cross-validation iterations.



(a) RADAM with ConvNeXt-XL.

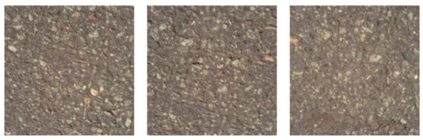


(b) GAP with ConvNeXt-XL.

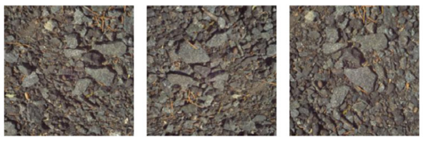


(c) Most challenging class for RADAM: "rust cover", often misclassified as "aluminum".

(d) Samples from the "aluminum" class.

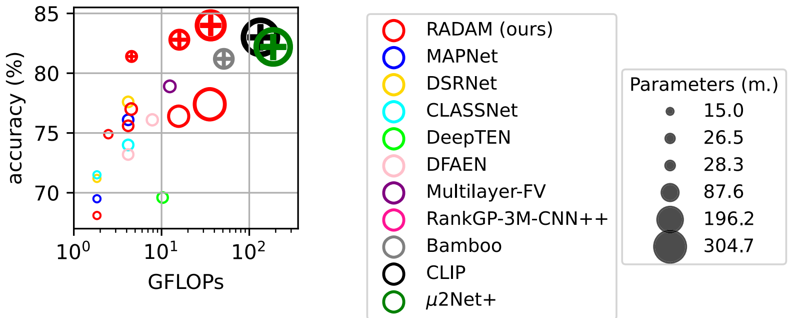


(e) Samples from the class "asphalt-stone".

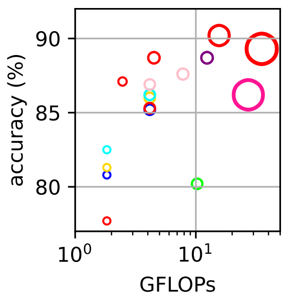


(f) Samples from the class "stone-asphalt".

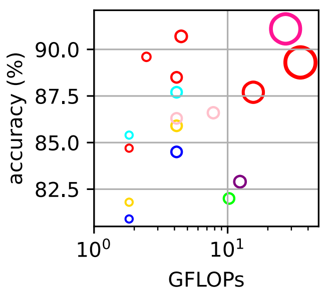
Fig. 5. Confusion matrix and image samples from the most challenging classes of the GTOS benchmark. The confusion matrix is normalized by the total number of samples per class after the cross-validation iterations.



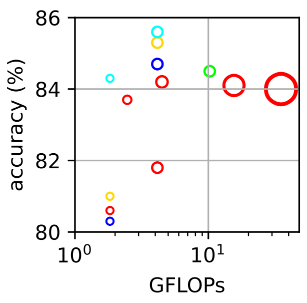
(a) DTD.



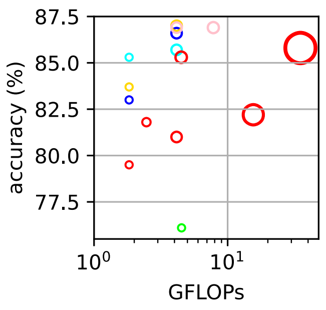
(b) FMD.



(c) KTH-TIPS2-b.



(d) GTOS.



(e) GTOS-Mobile.

Fig. 6. Computational budget at inference time according to the backbone used by different methods. The symbol ⊕ represents when stronger pre-training was used (with datasets bigger than ImageNet-1k).

chosen dataset or the fine-tuning of the methods that do so. In this sense, we show in Table 5 the approximate inference costs of the different backbones employed by the methods that are explored in this work, including the proposed RADAM.

We evaluate the relationship between the cost and performance of the methods according to the backbones they employ, and the results are shown in Fig. 6 for all the datasets. For RADAM, we consider the backbones ResNet18, ConvNeXt-nano, ResNet50, ConvNeXt-T, ConvNeXt-B, and ConvNeXt-L (in this order according to the x-axis of the plots). It is possible to notice the superiority of RADAM at different budgets on DTD, FMD, and KTH. Moreover, the performance scales with backbone complexity and better pre-training (DTD).

Multilayer-FV achieved the previous state-of-the-art on FMD using EfficientNet-B5 with an input size of 512 pixels, which yields an approximate inference cost of 12 GFLOPs. This is considerably higher than the cost of ConvNeXt-T (4.5 GFLOPs), with which RADAM achieves an improvement of 4.3% in absolute performance compared to Multilayer-FV. On KTH, RankGP-3M-CNN++ achieves 91.1% accuracy (previous state-of-the-art) using three backbones, with a total inference cost of around 30 GFLOPs, while RADAM with ConvNeXt-T achieves comparable results (−0.1%), and also achieves the state-of-the-art results using ConvNeXt-B (91.8% with at 15 GFLOPs), and ConvNeXt-XL (94.4% at 61 GFLOPs). Considering GTOS and GTOS-M, competitive cost and performance are also achieved with RADAM using ConvNeXt-T.

To increment the cost analysis, we show in Table 6 the practical running time of RADAM (with $m = 4$ and SVM) in contrast to the costs of ResNet and other pooling techniques (GAP, GMP, and GLpP). Our intuition is to compare RADAM to different feature extraction techniques, and to the approach in previous works of fine-tuning the backbone. We measure the average time (in milliseconds) of 100 runs of ResNet18 and ResNet50 over one 224×224 RGB image considering the RADAM module or pooling techniques, the ResNet forward and backward passes, and the corresponding times when fine-tuning the backbone or using feature extraction with SVM for inference. It is important to notice that the cost of RADAM varies according to the backbone since the number of

Table 5
Computational budget of different backbones employed by texture recognition methods explored in this work.

Backbone	Input size	GFLOPs	Parameters (millions)
ResNet18	224 × 224	1.8	11.2
ConvNeXt-nano	224 × 224	2.4	14.9
ResNet50	224 × 224	4.1	23.5
ConvNeXt-T	224 × 224	4.5	27.8
ResNet18	352 × 352	4.5	11.2
DenseNet161	224 × 224	7.8	26.5
ResNet50	352 × 352	10.2	23.5
EfficientNet-B5	512 × 512	12.3	28.3
ConvNeXt-B	224 × 224	15.4	87.6
ConvNeXt-L	224 × 224	34.4	196.2
ViT-B/16	224 × 224	49.4	86.1
ConvNeXt-XL	224 × 224	60.9	348.1
ViT-L/14	336 × 336	125.0	304.3
ViT-L/16	384 × 384	174.8	304.7

Table 6
Average running time (in milliseconds) of 100 repetitions using a 224 × 224 RGB image, performed on a machine with a GTX 1080ti, Intel Core i7-7820X 3.60 GHz processor (using 8 threads), and 64 GB of RAM.

		Time (ms)	
Method	Backbone	CPU	GPU
backbone's forward pass	ResNet18	15.0±1.6	4.3±0.1
backbone's backward pass	ResNet18	50.8±5.3	9.9±1.5
backbone's 1 fine-tuning epoch	ResNet18	65.8±6.0	14.2±1.6
GAP	ResNet18	0.4±0.03	0.2±0.01
GMP	ResNet18	0.4±0.02	0.2±0.01
GLpP	ResNet18	0.3±0.02	0.2±0.02
RADAM	ResNet18	2.7±0.2	2.9±0.04
backbone + GAP + SVM inference	ResNet18	15.9±2.1	6.8±0.2
backbone + GMP + SVM inference	ResNet18	16.4±1.8	10.5±0.2
backbone + GLpP + SVM inference	ResNet18	15.6±1.7	6.6±4.1
backbone + RADAM + SVM inference	ResNet18	19.8±1.6	8.1±0.1
backbone's forward pass	ResNet50	34.0±2.6	11.1±0.8
backbone's backward pass	ResNet50	107.6±5.9	21.5±2.2
backbone's 1 fine-tuning epoch	ResNet50	141.7±7.6	32.6±2.3
GAP	ResNet50	0.6±0.02	0.2±0.01
GMP	ResNet50	0.8±0.1	0.2±0.01
GLpP	ResNet50	0.4±0.03	0.2±0.02
RADAM	ResNet50	7.0±0.4	5.2±0.4
backbone + GAP + SVM inference	ResNet50	35.8±3.9	15.2±0.4
backbone + GMP + SVM inference	ResNet50	35.4±3.0	19.5±1.3
backbone + GLpP + SVM inference	ResNet50	33.5±2.9	13.6±0.9
backbone + RADAM + SVM inference	ResNet50	50.3±2.6	18.1±0.8

aggregate features (z) changes, thus impacting the RAE training cost.

Considering the results in Table 6, fine-tuning ResNet50 on GTOS-M using only 10 epochs with a batch size of 1 would take around 40 h on CPU or 9 h on GPU. On the other hand, extracting features with RADAM followed by SVM inference on the whole GTOS-M dataset takes around 1.3 h on CPU or half an hour on GPU. The training of SVM on the whole dataset (on CPU) takes an additional 15 min on average, without hyperparameter tuning. The results demonstrate that the RADAM is considerably faster than the ResNet backbone in terms of training/fine-tuning. Although the RADAM module alone is usually slower than other pooling techniques, which is to be expected since they perform simpler operations, the differences are negligible when considering the full inference pipeline with SVM. Moreover, RADAM provides considerably better performance than these pooling techniques (see Table 2). These results extend to ConvNeXt-nano and ConvNeXt-T, considering that the cost is comparable to ResNet18 and ResNet50, respectively. In summary, these findings corroborate our claims that RADAM provides both considerable savings in training time (compared to fine-tuning) while achieving SOTA results at similar inference costs.

5. Conclusion

We presented RADAM, a new feature encoding module for texture analysis. The method consists of randomly encoding aggregated deep activation maps from pre-trained DCNN using RAEs. These autoencoders learn to pool activation maps into a 1-dimensional representation by training on its z-dimentional pixels as sample points. A texture image is then encoded by using the decoder weights learned from its activation maps. The procedure is orderless but takes into account the spatial information of the pixels by using a 2D positional encoding. Compared to previous works, our method does not require fine-tuning of the backbone, and the encoding module is rather simple. Linear classification of the descriptors is performed with an SVM, and we achieve state-of-the-art performance on several texture benchmarks. RADAM also achieves the best efficiency considering inference cost and performance using backbones with varying computational budgets. These results are impressive also considering that, compared to other methods, no fine-tuning of the backbone is needed for RADAM, causing a lower cost also at training time.

Our work corroborates a simpler approach to texture recognition where the fine-tuning of costly backbones may not be necessary to achieve high discriminatory power. For future works, one may explore different backbones or different formulations of our RAE, with multiple layers, more hidden neurons, and other possible improvements. On the other hand, if enough computing resources are available, another approach more similar to previous works would be to explore our module in an end-to-end manner. Since RADAM is deterministic and a closed-form solution, an alternative would be adding a linear layer instead of an SVM and optimizing it along with the backbone.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The code for RADAM is open source and is available on GitHub through the link informed in the paper.

Acknowledgements

L. Scabini acknowledges funding from the São Paulo Research Foundation (FAPESP) (Grants #2019/07811-0 and #2021/09163-6) and the National Council for Scientific and Technological Development (CNPq) (Grant #142438/2018-9). K M. Zielinski acknowledges support from FAPESP (Grant #2022/03668-1) and the Higher Education Personnel Improvement Coordination (CAPES) (Grant #88887.631085/2021-00). L. C. Ribas acknowledges support from FAPESP (Grants #2021/07289-2 and #2018/22214-6). W. N. Gonçalves acknowledges support from CNPq (Grants #405997/2021-3 and #305296/2022-1). O. M. Bruno acknowledges support from CNPq (Grant #305610/2022-8) and FAPESP (grants #2018/22214-6 and #2021/08325-2). The authors are also grateful to the NVIDIA GPU Grant Program. This research received funding from the Flemish Government under the "Onderzoeksprogramma Artificile Intelligentie (AI) Vlaanderen" programme.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at 10.1016/j.patcog.2023.109802.

References

[1] L.F. Scabini, L.C. Ribas, O.M. Bruno, Spatio-spectral networks for color-texture analysis, Inf. Sci. 515 (2020) 64–79.
[2] T. Ojala, M. Pietikainen, T. Maenpaa, Multiresolution gray-scale and rotation invariant texture classification with local binary patterns, IEEE Trans. Pattern Anal. Mach. Intell. 24 (7) (2002) 971–987.
[3] R. Azencott, J.-P. Wang, L. Younes, Texture classification using windowed Fourier filters, IEEE Trans. Pattern Anal. Mach. Intell. 19 (2) (1997) 148–153.
[4] A.R. Backes, D. Casanova, O.M. Bruno, Color texture analysis based on fractal descriptors, Pattern Recognit. 45 (5) (2012) 1984–1992.
[5] L.F. Scabini, R.H. Condori, W.N. Gonçalves, O.M. Bruno, Multilayer complex network descriptors for color–texture characterization, Inf. Sci. 491 (2019) 30–47.
[6] J. Zhang, T. Tan, Brief review of invariant texture analysis methods, Pattern Recognit. 35 (3) (2002) 735–747.
[7] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, A. Vedaldi, Describing textures in the wild, in: 2014 IEEE Conference on Computer Vision and Pattern Recognition, 2014, pp. 3606–3613.
[8] Z. Yang, S. Lai, X. Hong, Y. Shi, Y. Cheng, C. Qing, Dfaen: double-order knowledge fusion and attentional encoding network for texture recognition, Expert Syst. Appl. 209 (2022) 118223.
[9] W. Zhai, Y. Cao, Z.-J. Zha, H. Xie, F. Wu, Deep structure-revealed network for texture recognition, in: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020, pp. 11007–11016.
[10] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, Extreme learning machine: theory and applications, Neurocomputing 70 (1) (2006) 489–501.
[11] Y.-H. Pao, Y. Takefuji, Functional-link net computing: theory, system architecture, and functionalities, Computer 25 (5) (1992) 76–79.
[12] W.F. Schmidt, M.A. Kraaijveld, R.P.W. Duin, Feedforward neural networks with random weights, in: Proceedings., 11th IAPR International Conference on Pattern Recognition. Vol.II. Conference B: Pattern Recognition Methodology and Systems, 1992, pp. 1–4.
[13] L.L.C. Kasun, H. Zhou, G.-B. Huang, C.M. Vong, Representational learning with elms for big data, IEEE Intell. Syst. 28 (2013) 31–34.
[14] L. Liu, J. Chen, P. Fieguth, G. Zhao, R. Chellappa, M. Pietikäinen, From BoW to CNN: two decades of texture representation for texture classification, Int. J. Comput. Vis. 127 (1) (2018) 74–109.
[15] M. Cimpoi, S. Maji, A. Vedaldi, Deep filter banks for texture recognition and segmentation, in: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 3828–3836.
[16] R.H. M. Condori, O.M. Bruno, Analysis of activation maps through global pooling measurements for texture classification, Inf. Sci. 555 (2021) 260–279.
[17] L.O. Lyra, A.E. Fabris, J.B. Florindo, Multilayer deep feature extraction for visual texture recognition, 2022.
[18] H. Zhang, J. Xue, K. Dana, Deep ten: texture encoding network, in: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE Computer Society, Los Alamitos, CA, USA, 2017, pp. 2896–2905.
[19] J. Xue, H. Zhang, K. Dana, Deep texture manifold for ground terrain recognition, in: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, IEEE, 2018.
[20] T.-Y. Lin, A. RoyChowdhury, S. Maji, Bilinear CNN models for fine-grained visual recognition, in: 2015 IEEE International Conference on Computer Vision (ICCV), 2015, pp. 1449–1457.
[21] J. Xue, H. Zhang, K. Nishino, K. Dana, Differential viewpoints for ground terrain material recognition, IEEE Trans. Pattern Anal. Mach. Intell. 44 (2020) 1205–1218.
[22] W. Zhai, Y. Cao, J. Zhang, Z.-J. Zha, Deep multiple-attribute-perceived network for real-world texture recognition, in: 2019 IEEE/CVF International Conference on Computer Vision (ICCV), 2019, pp. 3612–3621.
[23] Z. Chen, F. Li, Y. Quan, Y. Xu, H. Ji, Deep texture recognition via exploiting cross-layer statistical self-similarity, in: 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2021, pp. 5227–5236.
[24] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, N. Houlsby, An image is worth 16*x*16 words: transformers for image recognition at scale, CoRR abs/2010.11929(2020). https://arxiv.org/abs/2010.11929.
[25] A. Radford, J.W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al., Learning transferable visual models from natural language supervision, in: International Conference on Machine Learning, PMLR, 2021, pp. 8748–8763.
[26] Y. Zhang, Q. Sun, Y. Zhou, Z. He, Z. Yin, K. Wang, L. Sheng, Y. Qiao, J. Shao, Z. Liu, Bamboo: building mega-scale vision dataset continually with human-machine synergy, 2022.
[27] A. Gesmundo, A continual development methodology for large-scale multitask dynamic ML systems, arXiv preprint arXiv:2209.07326(2022).
[28] J.J.M. Sá Junior, A.R. Backes, ELM based signature for texture classification, Pattern Recognit. 51 (2016) 395–401.
[29] L.C. Ribas, J.J.M. Sá Junior, L.F. Scabini, O.M. Bruno, Fusion of complex networks and randomized neural networks for texture analysis, Pattern Recognit. 103 (2020) 107189.
[30] E.H. Moore, On the reciprocal of the general algebraic matrix, Bull. Am. Math. Soc. 26 (1920) 394–395.
[31] R. Penrose, A generalized inverse for matrices, Math. Proc. Camb. Philos. Soc. 51 (3) (1955) 406–413.
[32] A.M. Saxe, J.L. McClelland, S. Ganguli, Exact solutions to the nonlinear dynamics of learning in deep linear neural networks, arXiv preprint arXiv:1312.6120(2013).
[33] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision And pattern Recognition, 2016, pp. 770–778.
[34] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, S. Xie, A convnet for the 2020s, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 11976–11986.
[35] R. Wightman, Pytorch image models, 2019, https://github.com/rwightman/pytorch-image-models.
[36] Z. Wang, J.-C. Liu, Translating math formula images to latex sequences using deep neural networks with sequence-level training, Int. J. Doc. Anal. Recognit. (IJDAR) 24 (1) (2021) 63–75.
[37] M. Wortsman, G. Ilharco, S.Y. Gadre, R. Roelofs, R. Gontijo-Lopes, A.S. Morcos, H. Namkoong, A. Farhadi, Y. Carmon, S. Kornblith, et al., Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time, in: International Conference on Machine Learning, PMLR, 2022, pp. 23965–23998.
[38] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, PyTorch: an imperative style, high-performance deep learning library, in: Advances in Neural Information Processing Systems, vol. 32, Curran Associates, Inc., 2019, pp. 8024–8035.
[39] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: machine learning in Python, J. Mach. Learn. Res. 12 (2011) 2825–2830.
[40] S. Huovinen, M. Pietikinen, T. Ojala, J. Kyllnen, J. Viertola, T. Menp, Outex - new framework for empirical evaluation of texture analysis algorithms, in: Proceedings of 16th International Conference on Pattern Recognition, vol. 1, IEEE Computer Society, Los Alamitos, CA, USA, 2002, p. 10701.
[41] L. Sharan, R. Rosenholtz, E. Adelson, Material perception: what can you see in a brief glance? J. Vis. 9 (2010) 784.
[42] B. Caputo, E. Hayman, P. Mallikarjuna, Class-specific material categorisation, in: Tenth IEEE International Conference on Computer Vision (ICCV'05), vol. 1, 2, 2005, pp. 1597–1604.
[43] B.D. Ripley, Pattern Recognition and Neural Networks, Cambridge University Press, 2007.
[44] J. Platt, et al., Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods, Adv. Large Margin Classif. 10 (3) (1999) 61–74.
[45] L. Breiman, Random forests, Mach. Learn. 45 (2001) 5–32.
[46] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, ImageNet: a large-scale hierarchical image database, in: 2009 IEEE Conference on Computer Vision and Pattern Recognition, IEEE, 2009, pp. 248–255.

Leonardo Scabini is a Ph.D. candidate in Computational Physics at the São Carlos Institute of Physics of the University of São Paulo, where he also received an M.Sc. degree in Computational Physics (2018). He has a B.Sc. in Computer Science from the Federal University of Mato Grosso do Sul (2015), and from January 2022 to January 2023 he was an intern at the Knowledge-Based Systems (KERMIT) research unit at Ghent University, Belgium. His research interests include computer vision, network science, neural networks, and their applications

Kallil Zielinski has received a B.Sc. in Computer Engineering (2018) at the Federal Technological University of Paraná, and his M.Sc in Electrical Engineering (2022) also at the Federal Technological University of Paraná. Currently, he is a Ph.D. candidate at IFSC/USP. His fields of interest are Computer Vision, Neural Networks, Network Sciences, and Reinforcement Learning

Lucas C. Ribas received a B.Sc. (2014) in Information Systems at the Federal University of Mato Grosso do Sul, an M.Sc. (2017) in Computer Science at the Institute of Mathematics and Computer Science at the University of S. Paulo (ICMC/USP) and a Ph.D. (2021) in Computer Sciences at the ICMC/USP. Currently, he is a Professor at the Institute of Biosciences, Humanities, and Exact Sciences at the São Paulo State University. His fields of interest include Computer Vision, Neural Networks, Complex Networks, and Pattern Recognition

Wesley Nunes Gonçalves received the B.Sc. degree in Computer Engineering from Dom Bosco Catholic University, Brazil, in 2007, the M.Sc. degree in Computer Science from the University of São Paulo, Brazil, in 2010, and the Ph.D. degree in Computational Physics from University of São Paulo, Brazil, in 2013. He is currently a Professor at the Federal University of Mato Grosso do Sul, Brazil, and is the author of several refereed papers. His topics of research include texture analysis, computer vision, complex networks, and the application of computational methods in precision agriculture

Bernard De Baets received an M.Sc. degree in Mathematics, a Postgraduate Degree in Knowledge Technology, and a Ph.D. degree in Mathematics from Ghent University, Belgium, in 1988, 1991, and 1995, respectively. He is currently a Senior Full Professor at Ghent University, where he is leading the research unit Knowledge-Based Systems (KERMIT) as well as the department of Data Analysis and Mathematical Modelling. He has acted as supervisor of 86 Ph.D. students and has published over 600 peerreviewed journal papers. He has delivered over 300 (invited) conference lectures. At present, he is Co-Editor-in-Chief of Fuzzy Sets and Systems and is a member of the Editorial Board of several other journals. He is a Government of Canada Award holder, has been nominated for the Ghent University Prometheus Award for Research, is a Fellow of the International Fuzzy Systems Association, a recipient of the EUSFLAT Scientific Excellence Award, an Honorary Professor of Budapest Tech, an Honorary Member of EUSFLAT, a Doctor Honoris Causa of the University of Turku, a Profesor Invitado of the Universidad Central "Marta Abreu" de Las Villas in Cuba and a Professor Extraordinarius at the University of South Africa

Odemir Bruno is a Professor at the São Carlos Institute of Physics at the University of São Paulo in Brazil. He is head of the Scientific Computing Group. He received his B.Sc. in Computer Science, his Ph.D. in Physics at the University of S. Paulo (Brazil), and his habilitation in Computer Science at the University of S. Paulo (Brazil). He is the author of about two hundred papers in journals and several book chapters. He is a co-author of two books and an inventor of seven patents. His fields of interest include Artificial Intelligence, Computer Vision, Image Analysis, Chaos, Fractals, Complex Systems, Network Science, Pattern Recognition, Plant Sciences, and Bioinformatics