Transfer Learning between Texture Classification Tasks using Convolutional Neural Networks

Luiz G. Hafemann, Luiz S. Oliveira
Department of Informatics
Federal University of Parana
Curitiba, PR, Brazil
luiz.gh@gmail.com, lesoliveira@inf.ufpr.br

Paulo R. Cavalin
IBM Research - Brazil
Rio de Janeiro, RJ, Brazil
pcavalin@br.ibm.com

Robert Sabourin
Lab. d'imagerie, de vision et d'intelligence artificielle
École de technologie supérieure
Université du Québec, Montreal, Canada
robert.sabourin@etsmtl.ca

***Abstract*—Convolutional Neural Networks (CNNs) have set the state-of-the-art in many computer vision tasks in recent years. For this type of model, it is common to have millions of parameters to train, commonly requiring large datasets. We investigate a method to transfer learning across different texture classification problems, using CNNs, in order to take advantage of this type of architecture to problems with smaller datasets. We use a Convolutional Neural Network trained on a source dataset (with lots of data) to project the data of a target dataset (with limited data) onto another feature space, and then train a classifier on top of this new representation. Our experiments show that this technique can achieve good results in tasks with small datasets, by leveraging knowledge learned from tasks with larger datasets. Testing the method on the the Brodatz-32 dataset, we achieved an accuracy of 97.04% - superior to models trained with popular texture descriptors, such as Local Binary Patterns and Gabor Filters, and increasing the accuracy by 6 percentage points compared to a CNN trained directly on the Brodatz-32 dataset. We also present a visual analysis of the projected dataset, showing that the data is projected to a space where samples from the same class are clustered together - suggesting that the features learned by the CNN in the source task are relevant for the target task.**

I. Introduction

Texture classification is an important task in image processing and computer vision, with a wide range of applications, such as computer-aided medical diagnosis [1], [2], [3], classification of forest species [4], [5], [6], classification in aerial/satellite images [7] and writer identification and verification [8]. This problem is commonly treated as a pattern recognition problem, and many feature descriptors have been developed targeting this particular type of problem, as reviewed by Zhang et al. [9] and tested on multiple datasets by Guo et al. [10]. Noteworthy techniques are: Gray-Level Co-occurrence Matrices (GLCM), the Local Binary Pattern operator (LBP) and Gabor filters.

In recent years, there has been an increased interest on machine learning models that do not rely on hand-engineered feature extractors, but that instead use *raw* data and learn the representations. This is the case of Deep Learning models, as reviewed by Bengio in [11] and [12]. From these models, Convolutional Neural Networks have been used to set the state-of-the-art in many computer vision tasks, such as object classification (as in the CIFAR dataset [13]), localization and object classification over a large number of classes (an in the ImageNet dataset [14]). This type of model has been particularly successful on large datasets, such as the ImageNet dataset that contains 1.2 million images on 1000 categories. For such tasks, it is common to have a large number of trainable parameters (in the order of millions), requiring a significant amount of computing power to train the network.

In the task of texture classification, CNNs are not yet widely used. Titive et al. [15] used convolutional neural networks on the Brodatz texture dataset, but considered only low resolution images, and a small number of classes. More recently, Hafemann et al. tested this type of model for Forest Species classification with good results [16]. For forest species recognition, the datasets were large enough to successfully train the network - for example, one of the datasets consists of 3k images of size 3264 x 2448. Considering the results from both object recognition and texture classification, having large datasets seem to be required for successfully training CNNs. In practical applications, however, there are several cases where we must deal with relatively small datasets. For this reason, techniques that try to overcome this situation, such as transfer learning, are important to increase the accuracy of such systems.

In this paper, we consider the task of transferring learning between texture classification tasks. Instead of training CNNs directly for a task (e.g. a task with a small dataset), we present and test a simple transfer learning method using Convolutional Neural Networks. A CNN is trained in a source task and it is used to project the data of a target dataset onto another feature space. The expectation is that the patterns learned on the source dataset can be used to project the samples of the target dataset onto a feature space where the classes are more easily separable. In practice, we use the CNN trained on the source dataset as a "feature extractor" for the target dataset. This idea is especially interesting to try to improve classification rates on tasks that have small datasets, where directly training CNNs may not perform well.

This strategy was inspired by the work of Oquab et. al [17], that used a CNN trained on the ImageNet dataset to classify images on a smaller dataset (Pascal VOC dataset) with improved results. Differently from their work, instead of copying the weights from the source neural network to a target network with additional layers, we use the source network to project the data to a new feature space, by performing forward propagation of the target dataset data through the network, obtaining a new representation for each sample. We then train another classifier (in this work, Support Vector Machines) on this new representation. We also adapt the training and

testing procedure to take advantage of particularities of texture datasets, extending the strategy for training CNNs on texture datasets presented in [16] to transfer knowledge across tasks.

This paper is organized as follows: section II reviews the foundations for transfer learning; section III presents our method for transferring learning between texture classification tasks, and the results are shown in section IV. Finally, section V presents the conclusions.

II. Brief Review on Transfer Learning

Transfer learning is a strategy that aims to improve the performance on a machine learning task by leveraging knowledge obtained from a different task. The intuition is: we expect that by learning one task we gain knowledge that could be used to learn similar tasks. For example, learning to recognize apples might help in recognizing pears, or learning to play the electronic organ may help learning the piano [18].

In a survey on transfer learning, Pan and Yang [18] use the following notations and definitions for transfer learning: A **domain** $\mathcal{D}$ consists of two components: a **feature space** $\mathcal{X}$ and a **probability distribution** $P(X)$, where $X = \{x_1, ..., x_n\} \in \mathcal{X}$. Given a specific domain, $\mathcal{D} = \{\mathcal{X}, P(X)\}$, a **task** consist of two components: a label space $\mathcal{Y}$ and an objective predictive function $f(.)$, denoted by $\mathcal{T} = \{\mathcal{Y}, f(.)\}$), which is not observed, but that can be learned from training data. The function $f(.)$ can then be used to predict the corresponding label, $f(x)$ of a new instance x. From a probabilistic perspective, $f(x)$ can be written as $P(y|x)$. In an example in the texture classification domain, we use a feature extractor on the textures to generate the feature space $\mathcal{X}$. A given training sample is denoted by X, $X \in \mathcal{X}$, with associated label y, $y \in \mathcal{Y}$. A model is then trained on a dataset to learn the function $f(.)$ (or $P(y|x)$).

Transfer learning can then be defined as follows:

Definition 2.1: Given a source domain $\mathcal{D}_S$ and learning task $\mathcal{T}_S$, a target domain $\mathcal{D}_T$ and learning task $\mathcal{T}_T$, transfer learning aims to help improve the learning of the target predictive function $f_T(.)$ in $\mathcal{D}_T$ using the knowledge in $\mathcal{D}_S$ and $\mathcal{T}_S$, where $\mathcal{D}_S \neq \mathcal{D}_T$, or $\mathcal{T}_S \neq \mathcal{T}_T$

To illustrate, we can use this definition for texture classification. We can apply transfer learning by using a source task from a different domain $\mathcal{D}$, for example, using a different feature space $\mathcal{X}$ (with a different feature descriptor) or using a dataset with different marginal probabilities $P(X)$ (i.e. textures that display different properties from the target task). We can also apply transfer learning if the task $\mathcal{T}$ is different: for example, using a source task that learns different labels $\mathcal{Y}$ or with different objective $f(.)$.

There are multiple categories of transfer learning. In the present work, we focus on *inductive transfer learning*, where we need labeled datasets for both the source and target tasks. In particular, we consider transferring knowledge of Feature Representations, where the basic idea is to learn a low-dimensional representation on the source task, that is shared across related tasks.

A. Transfer Learning in Convolutional Neural Networks

The architecture of state-of-the-art Convolutional Neural Networks often contains a large number of parameters, reaching the order of millions. Learning so many parameters may present poor results with small datasets. Transfer learning can assist in this task, by using the internal representation learned from one task, and applying it to another task. As noted by Oquab et. al [17], a CNN can act as a generic extractor of mid-level representations, which can be pre-trained in a source task, and then re-used in other target tasks.

One difficulty to apply this method is that the distribution of the images, $P(X)$, may differ significantly between the source and target tasks. Oquab et. al [17] propose a simple method to cope with this problem, by training an adaptation layer: after the CNN is trained on the source task, the parameters learned by network, except for the last layer, are transferred to the target task. Two fully-connected layers are added to the network, and trained with the training set from the target task. Using this simple transfer learning procedure, Oquab et. al were able to achieve state-of-the-art results on the competitive Pascal VOC dataset.

III. Proposed Method

We now present a method for transferring learning across different texture classification tasks. This method has been inspired by the work of Oquab et al. [17] on transfer learning for object recognition and adapted to texture classification, taking advantage of particular characteristics of textures. From a high-level, we follow these steps:

- Train a CNN in the source task
- Use this network to obtain a new representation of the target task (project the data to another feature space)
- Use this new representation to train a model

The following sections present the details of the method.

A. CNN training

We train a Convolutional Neural Network on the source task following the procedure presented in [16]. This method is described in Algorithm 1, and essentially performs Stochastic Gradient Descent to update the weights of the CNN, using, in each epoch, a different random patch from each texture in the training set. This procedure is particularly useful for textures, where it is often possible to classify a sample with a small region of the image. We also found that in practice this method helps preventing overfitting the training set.

This architecture is based on [16], and consists of the following layers and parameters:

1) *Input layer*: patches from the original image, of size s_p x s_p
2) *Two combinations of convolutional and pooling layers*: each convolutional layer has 64 filters and stride set to 1. The first convolution layer has filters of size s_{f1} x s_{f1} and the second convolution uses filters of size 3x3. The pooling layers consist of windows with size 3x3 and stride 2;
3) *Two locally-connected layers*: 32 filters of size 3x3 and stride 1;
4) *Fully-connected output layer*: dependent on the number of classes of the problem. In our case, 41 classes on the source dataset.

Algorithm 1 Training with Random Patches

Require: dataset, patchsize, batch_size, learning_rate, momentum_factor

model_state ← *random weights*
repeat
  dataset$_{epoch}$ ← *empty list*
  for each image in dataset **do**
    Insert(dataset$_{epoch}$, **Random_Image_Crop**(image, patchsize))
  end for
  numBatches ← size(dataset) / batch_size
  for batch ← 0 **To** numBatches **do**
    dataset$_{batch}$ ← dataset$_{epoch}$[batch * batch_size : (batch+1) * batch_size -1]
    model_state ← **ForwardProp**(model, dataset$_{batch}$.X)
    gradients ← **BackProp**(model_state, dataset$_{batch}$.Y)
    ApplyGradients(model, gradients, learning_rate, momentum_factor)
  end for
until Convergence_Criteria()

These hyperparameters were selected according to [16], and we optmized the values for the input layer (size of the patches, s_p) and the size of the filters from the first convolutional layer (s_{f1}) using a grid search. We considered inputs of size 32x32, 48x48, 64x64 and 96x96; and filters of size 3x3, 5x5, 7x7, 10x10 and 12x12.

B. Training a model for Transfer learning

After training a CNN on the source task, we use it to obtain a new representation for the target dataset. One interpretation for this procedure is to consider that the source task (learn a CNN model on a texture dataset) learns feature extractors that are generic, to some extent, and that are useful for the target task. We use the CNN to extract a new representation for the target dataset, which is similar to using a feature extractor to the input, obtaining a new vector representation for each sample. When extracting the new representation, similar to Oquab et al. [17], we use the layers of the CNN (trained on the source task) up to the last layer before softmax.

When training the CNN using the method above, we use patches of the original images, instead of the full image. This means that when we generate a new representation on the target dataset, we are generating new representations for patches of the target images. Since SVM requires a static (fixed) dataset for training, we decided to consider two approaches to extract the patches from the target images: 1) using the set of non-overlapping patches (grid patches) and 2) Using N random patches from each image, where N is a hyperparamenter for the system. After extracting the patches from the images on the target dataset, we generate a new representation for the patches (using the CNN) obtaining a new dataset. We use this dataset for training an SVM. These steps are formalized in Algorithm 2 and illustrated in Figure 1.

C. Obtaining predictions on the target dataset

The test procedure using Transfer Learning is also adapted for textures. First, we extract the patches of the testing set, using the same approach used for the training set. We use the model trained on the source task to generate the new representation, and use the model trained using algorithm 2 to generate the predictions for each patch of the image. We then combine the results of the patches and calculate the accuracy. This procedure is described in Algorithm 3 and illustrated in Figure 2.

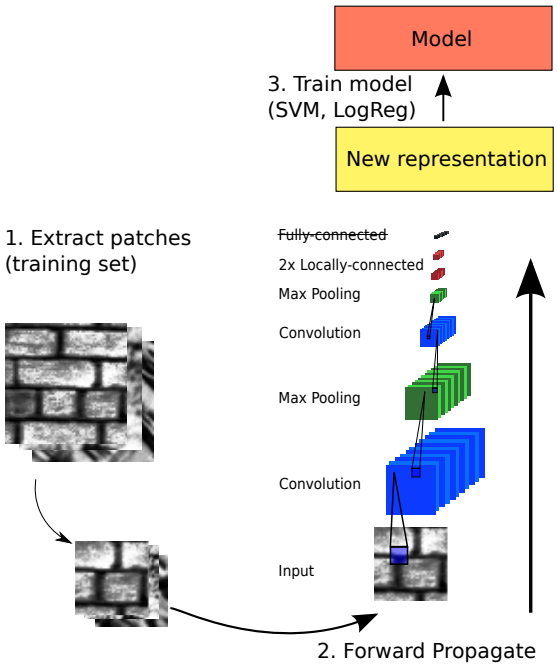

Fig. 1. The training procedure for transfer learning. First, we extract patches from the training set of the target task. For each patch, a new representation is obtained by forward-propagating the sample through the CNN trained on the source task. With this new dataset, a machine learning model is trained.

IV. Experiments

We now present the results of our experiments with Transfer Learning. For each experiment, we followed the procedure listed above to generate the new representations for the target dataset. We trained Support Vector Machines (SVMs) using the python package scikit-learn [19], that provides a wrapper over the libsvm [20] library. We trained SVMs with the radial basis function (RBF) kernel, selecting the hyperparameters C and γ using a grid search, following the recommended values from Hsu [21]. It is worth noting that due to limitations on

Algorithm 2 Transfer Learning - training

Require: $\text{dataset\_train}_{\text{source}}$, $\text{dataset\_train}_{\text{target}}$, $\text{dataset\_test}_{\text{target}}$, patchSize

$\text{model}_{\text{source}} \leftarrow$ **Train_With_Random_Patches**($\text{dataset\_train}_{\text{source}}$, patchSize)
$\text{Patches\_train}_{\text{target}} \leftarrow$ *empty list*
for each image in $\text{dataset\_train}_{\text{target}}$ **do**
    Insert($\text{Patches\_train}_{\text{target}}$, **ExtractPatches**(image, patchSize))
end for
$\text{NewDataset\_train}_{\text{target}} \leftarrow$ **GetNewRepresentation**($\text{model}_{\text{source}}$, $\text{Patches\_train}_{\text{target}}$)
$\text{Model}_{\text{target}} \leftarrow$ **TrainModel**($\text{NewDataset\_train}_{\text{target}}$)

Algorithm 3 Transfer Learning - test

Require: $\text{model}_{\text{source}}$, $\text{model}_{\text{target}}$, $\text{dataset\_test}_{\text{target}}$, patchSize

$\text{Patches\_test}_{\text{target}} \leftarrow$ *empty list*
for each image in $\text{dataset\_test}_{\text{target}}$ **do**
    Insert($\text{Patches\_test}_{\text{target}}$, **ExtractPatches**(image, patchSize))
end for
$\text{NewDataset\_test}_{\text{target}} \leftarrow$ **GetNewRepresentation**($\text{model}_{\text{source}}$, $\text{Patches\_test}_{\text{target}}$)
$\text{PatchProbabilities}_{\text{target}} \leftarrow$ **GetModelPredictions**($\text{model}_{\text{target}}$, $\text{NewDataset\_test}_{\text{target}}$)
ImageProbabilities $\leftarrow$ **CombineProbabilities**(PatchProbabilities)
Predictions $\leftarrow$ **argmax**(ImageProbabilities)
ClassifiedCorrectly $\leftarrow$ Count(Predictions = $\text{dataset\_test}_{\text{target}}$.Y)
Accuracy $\leftarrow$ ClassifiedCorrectly / Size(dataset)

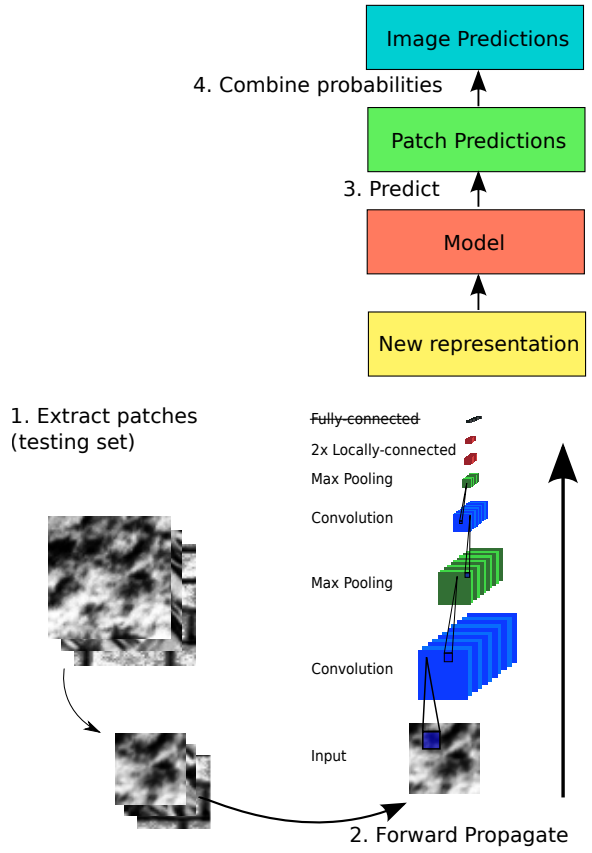

Fig. 2. The testing procedure for transfer learning. First, we extract patches from the testing set of the target task. For each patch, a new representation is obtained by forward-propagating the sample through the CNN trained on the source task. This new representation is used to obtain a prediction using the model trained for transfer learning. Lastly, the predictions of all patches of the image are combined.

computing power, we do not optimize these hyperparameters using the whole training set. Instead, a subset of five thousand examples is randomly selected to perform this optimization, and subsequently an SVM is trained with these hyperparameters on the whole training set. We performed the whole process using 3-fold cross validation, and report the mean and standard deviation of the accuracy.

A. *Datasets*

For this experiment, we selected one large texture dataset for training the source task, and a smaller texture dataset for the target task. For the source task, we used a dataset of macroscopic images for forest species recognition[22], the same dataset used in [16] to train a CNN. This dataset contains pictures of cross-section surfaces of the trees, obtained using a regular digital camera. This dataset contains 41 classes, and images with resolution 3264 x 2448. This dataset is large (around 65 GB in a bitmap format), which is desired for the source task. Examples from this dataset can be found in Figure 3

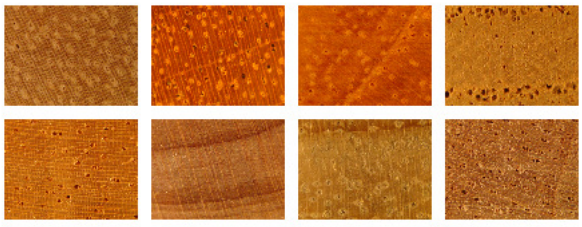

Fig. 3. Sample images from the macroscopic Brazilian forest species dataset

For the target task, we used a smaller dataset: Brodatz-32. The Brodatz album is a classical texture database, dating back to 1966, and it is widely used for texture analysis. [23]. This dataset is composed of 112 textures, with a single

640x640 image per each texture. In order to standardize the usage of this dataset for texture classification, Valkealahti [24] introduced the Brodatz-32 dataset, a subset of 32 textures from the Brodatz album, with 64 images per class (64x64 pixels in size), containing patches of the original images, together with rotated and scaled versions of them (see Figure 4). This dataset is small (around 8 MB in a bitmap format) and was selected to identify if we can increase classification performance on small datasets using transfer learning. Samples from this dataset can be found in Figure 4.

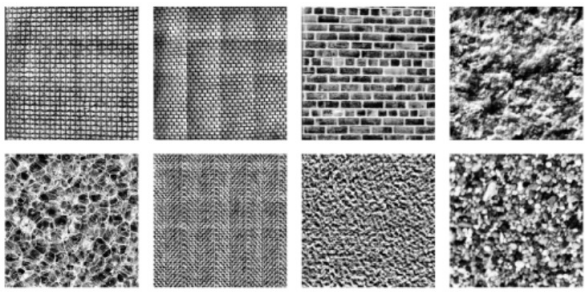

Fig. 4. Sample images from the Brodatz-32 dataset [24]

B. Training

We first trained a CNN on the source task using Algorithm 1, similarly to the work presented in [16]. Differently from this previous work, we first converted the images to grayscale before training the CNN, since the images from the target dataset (Brodatz) do not have colors. It is worth noting that we did not see a significant performance drop compared to the CNN trained using the original dataset (with colors). For training, we obtained best results using patches of size 32x32, and a filter size of 12x12 on the first convolutional layer. We kept the remaining parameters as reported in [16].

After training the CNN, we used it to obtain a new representation for the Brodatz-32 dataset. We first split the dataset, with half of the samples for training and half for testing, as in previous work on this dataset ([25], [26]). We then extracted 32x32 patches from the Brodatz-32 images, used the CNN to obtain a new representation, and trained an SVM classifier, following Algorithm 2. As described in section III, we considered two strategies for extracting the patches: the set of non-overlapping patches and random patches. In this case, the set of non-overlapping patches generated 4 patches of 32x32 pixels per image (each image of size 64x64). For random patches we tested multiple values, ranging from 2 patches to 80 patches per image.

C. Results

We first present the results of the tests with different patch extraction alternatives, as discussed above. Figure 5 shows the performance (classification accuracy) with different number of random patches extracted per image, comparing it to the result using the set of non-overlapping patches (grid patches). We can see that we obtain a superior performance using random patches (97.04%), but only when we consider a large number of patches per image. It is worth noting that this alternative requires more computational power (more samples to classify for a given image), that needs to be balanced with the simpler approach (non-overlapping patches) that achieved a performance of 96.35%.

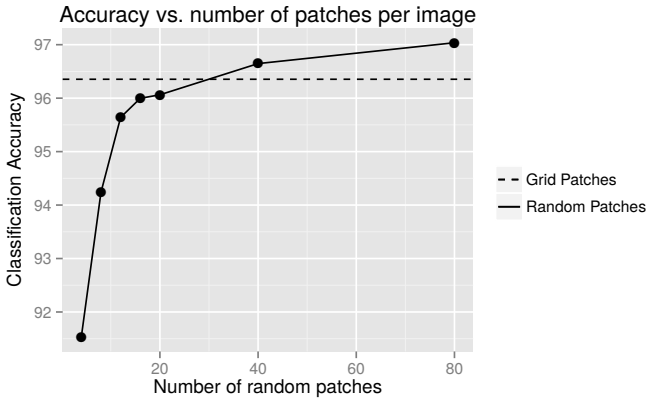

Fig. 5. Accuracy on the Brodatz-32 dataset using different number of Random Patches, compared to using Grid Patches.

Table I compares the method presented in this paper with the state-of-the art for classification in the Brodatz-32 dataset. We also present the results of a CNN trained in the Brodatz-32 dataset, according to [16]. We can see that the proposed method, leveraging the features learned on a larger dataset, was able to achieve results close to the state-of-the-art - 97.04% of accuracy compared to 98.90% of accuracy using Robust LBP. We can also see that this method present much better results than training a CNN directly on the Brodatz-32 dataset. This is likely due to the fact that the Brodatz-32 dataset is very small, and the CNN trained directly on it was unable to learn good representations. It is interesting to notice that even though the samples from the two datasets (Brodatz-32 and the Forest Species dataset) are very different, we can see that the representations learned by the CNN trained on the Forest Species dataset were relevant for the Brodatz-32 task. We explore this observation in the next section.

TABLE I. CLASSIFICATION ON THE BRODATZ-32 DATASET

Features and Algorithm	Accuracy
LBP (KNN) [25]	91.40%
Completed LBP (KNN) [25]	92.80%
Gabor Filters (KNN) [25]	95.80%
Shape Size Pattern Spectra (Decision Trees) [26]	96.50%
Dominant LBP (KNN) [25]	98.30%
Robust LBP (KNN) [25]	**98.90**%
CNN trained on Brodatz-32	91.27% (+- 0.53%)
Transfer Learning (SVM)	**97.04**% (+- 0.65%)

D. Visualizing the data

We now explore visually how a CNN trained on one dataset can be useful for other tasks.

For this purpose, we use the t-SNE algorithm [27]. T-SNE is a popular algorithm for visualizing high-dimensional data, by projecting the samples onto two dimensions retaining the local structure of the data.

Our objective is to see how well the weights learned by a CNN trained on one task can help in projecting the samples of another task onto a dimension where they are more easily separable. Below we report three visualizations of the Brodatz-32 dataset: the original dataset, the projection of the samples

through a CNN trained on the same dataset, and the projection of the samples through a CNN trained on the Forest Species dataset.

For the original dataset, we extract all non-overlapping patches from the images, and used t-SNE to reduce the raw data - pixels of the patches, of size 1024 (32x32) - to two dimensions. For the two other representations, we performed forward-propagation through the CNN using these patches, obtaining a new representation. We then used t-SNE to reduce this representation (1152 dimensions, in both CNNs) to two dimensions. Using this 2D representation, we built a scatter plot, representing the different classes with different colors. This representation allows us to see how the samples are clustered together in the high-dimensional space.

Figure 6 contains the visualization of the original dataset. We can see that the samples from different classes (colors) are mixed together, with no clear separation. This means that distances (such as euclidean distance) in the representation space using the original pixels are not very useful (we would expect a classifier that relies on these distances to perform poorly, such as KNN and linear classifiers).

Fig. 6. Visualization of patches (32x32) from the original Brodatz-32 dataset, reduced to 2D using t-SNE.

Figure 7 contains the visualization of the Brodatz-32 dataset transformed through a CNN trained on the same dataset. We can see that for several classes the samples get grouped together in the feature space. For example, the dark blue cluster in the bottom of the page refers to the Brodatz image *D104*.

Figure 8 contains the visualization of the Brodatz-32 dataset transformed through a CNN trained on the Forest Species dataset. It is interesting to note, that although the CNN had no access to samples of the Brodatz-32 dataset during training, it is still able to project the dataset to a feature space where samples from the same class get clustered together. We notice that in this case each class do not necessary form a single cluster: samples from the same class form multiple groups, but most of these groups are concise and separated from groups of other classes. This suggests that the features learned by the CNN trained on the Forest Species dataset are useful for the Brodatz-32 dataset, in spite of the large differences of the samples from the two datasets.

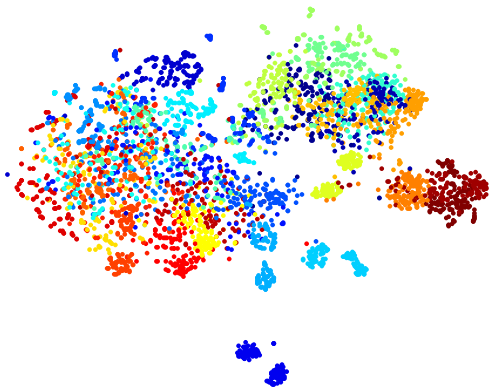

Fig. 7. New representation obtained by projecting the patches from the Brodatz dataset through a CNN (trained on the same dataset). The representation (1052 dimensions) is reduced to 2D using t-SNE for visualization.

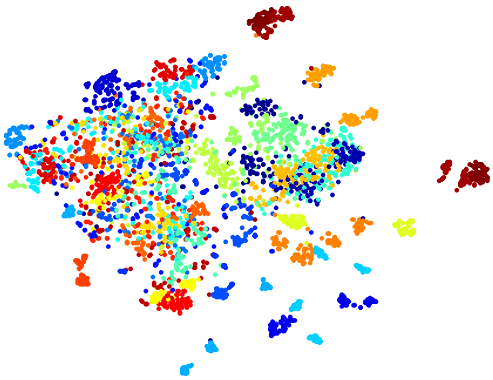

Fig. 8. New representation obtained by projecting the patches from the Brodatz dataset through a CNN trained on the Forest Species dataset. The representation (1052 dimensions) is reduced to 2D using t-SNE for visualization.

V. Conclusion

We presented a method to transfer learning across different texture classification problems, using CNNs. Our experiments showed that this method can be used to obtain high classification accuracy on tasks with limited amounts of data, by leveraging knowledge learned on tasks with large datasets, even if the tasks are not very similar. In the tests with the Brodatz-32 dataset, we obtained an accuracy of 97.04%, a result that is better than the classification accuracy of models that use some of the most popular texture descriptors, such as LBP (with 91.40% of accuracy) and Gabor Filters (with 95.80% of accuracy). This method also performed better than training a CNN directly on the Brodatz-32 dataset (which obtained an accuracy of 91.27%). We also presented a visual analysis of how the samples are projected to a better feature space by using CNNs, showing that the weights learned by the CNN in one task can be used to project the data of another dataset to a better feature space, where the samples are more easily separated.

References

[1] H. Harms, U. Gunzer, and H. M. Aus, "Combined local color and texture analysis of stained cells," *Computer Vision, Graphics, and Image Processing*, vol. 33, no. 3, pp. 364–376, Mar. 1986.

[2] A. Khademi and S. Krishnan, "Medical image texture analysis: A case study with small bowel, retinal and mammogram images," in *Canadian Conference on Electrical and Computer Engineering, 2008. CCECE 2008*, May 2008, pp. 001 949–001 954.

[3] R. Sutton and E. L. Hall, "Texture measures for automatic classification of pulmonary disease," *IEEE Transactions on Computers*, vol. C-21, no. 7, pp. 667–676, Jul. 1972.

[4] J. Y. Tou, Y. H. Tay, and P. Y. Lau, "A comparative study for texture classification techniques on wood species recognition problem," in *Natural Computation, 2009. ICNC'09. Fifth International Conference on*, vol. 5, 2009, pp. 8–12.

[5] P. Filho, L. Oliveira, A. Britto, and R. Sabourin, "Forest species recognition using color-based features," in *2010 20th International Conference on Pattern Recognition (ICPR)*, 2010, pp. 4178–4181.

[6] M. Nasirzadeh, A. Khazael, and M. bin Khalid, "Woods recognition system based on local binary pattern," in *2010 Second International Conference on Computational Intelligence, Communication Systems and Networks (CICSyN)*, 2010, pp. 308–313.

[7] R. Haralick, K. Shanmugam, and I. Dinstein, "Textural features for image classification," *IEEE Transactions on Systems, Man and Cybernetics*, vol. SMC-3, no. 6, pp. 610–621, Nov. 1973.

[8] D. Bertolini, L. S. Oliveira, E. Justino, and R. Sabourin, "Texture-based descriptors for writer identification and verification," *Expert Systems with Applications*, 2012.

[9] J. Zhang and T. Tan, "Brief review of invariant texture analysis methods," *Pattern recognition*, vol. 35, no. 3, pp. 735–747, 2002.

[10] Y. Guo, G. Zhao, and M. Pietikinen, "Discriminative features for texture description," *Pattern Recognition*, vol. 45, no. 10, pp. 3834–3843, Oct. 2012.

[11] Y. Bengio, "Learning deep architectures for AI," *Found. Trends Mach. Learn.*, vol. 2, no. 1, pp. 1–127, Jan. 2009.

[12] Y. Bengio and A. Courville, "Deep learning of representations," in *Handbook on Neural Information Processing*. Springer, 2013, pp. 1–28.

[13] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, "Maxout networks," arXiv e-print 1302.4389, Feb. 2013, JMLR WCP 28 (3): 1319-1327, 2013.

[14] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *CoRR*, vol. abs/1409.4842, 2014.

[15] F. H. C. Tivive and A. Bouzerdoum, "Texture classification using convolutional neural networks," in *TENCON 2006. 2006 IEEE Region 10 Conference*. IEEE, 2006, pp. 1–4.

[16] L. G. Hafemann, L. S. Oliveira, and P. Cavalin, "Forest species recognition using deep convolutional neural networks," in *Pattern Recognition (ICPR), 2014 22nd International Conference on*. IEEE, 2014, pp. 1103–1107.

[17] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, "Learning and transferring mid-level image representations using convolutional neural networks," in *CVPR*, 2014.

[18] S. J. Pan and Q. Yang, "A survey on transfer learning," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 22, no. 10, pp. 1345–1359, 2010.

[19] 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, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[20] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 27:1–27:27, 2011, software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

[21] C.-W. Hsu, C.-C. Chang, C.-J. Lin *et al.*, "A practical guide to support vector classification," 2003.

[22] P. L. P. Filho, L. S. Oliveira, S. Nisgoski, and A. S. Britto, "Forest species recognition using macroscopic images," *Machine Vision and Applications*, Jan. 2014.

[23] P. Brodatz, *Textures: a photographic album for artists and designers*. Dover New York, 1966, vol. 66.

[24] K. Valkealahti and E. Oja, "Reduced multidimensional co-occurrence histograms in texture classification," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 1, pp. 90–94, Jan. 1998.

[25] J. Chen, V. Kellokumpu, G. Zhao, and M. Pietikinen, "RLBP: Robust local binary pattern." BMVC, 2013.

[26] E. R. Urbach, J. B. Roerdink, and M. H. Wilkinson, "Connected shape-size pattern spectra for rotation and scale-invariant classification of gray-scale images," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 29, no. 2, pp. 272–285, 2007.

[27] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, no. 2579-2605, p. 85, 2008.