IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Date of publication October 2024, date of current version October 18, 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3484663

CG-CNN: Self-Supervised Feature Extraction Through Contextual Guidance and Transfer Learning

OLCAY KURSUN[1], AHMAD PATOOGHY[2], (SENIOR, IEEE), PEYMAN POURSANI[2], OLEG V. FAVOROV[3]

[1]Department of Computer Science, Auburn University at Montgomery, AL 36117, USA (e-mail: okursun@aum.edu)
[2]Department of Computer Systems Technology, North Carolina A&T State University, Greensboro, NC 27411, USA (e-mail: apatooghy@ncat.edu, pjpoursani@aggies.ncat.edu)
[3]Joint Department of Biomedical Engineering, University of North Carolina at Chapel Hill, NC, 27599 USA (e-mail: favorov@email.unc.edu)

Corresponding author: Ahmad Patooghy (e-mail: apatooghy@ncat.edu).

ABSTRACT Contextually Guided Convolutional Neural Networks (CG-CNNs) employ self-supervision and contextual information to develop transferable features across diverse domains, including visual, tactile, temporal, and textual data. This work showcases the adaptability of CG-CNNs through applications to various datasets such as Caltech and Brodatz textures, the VibTac-12 tactile dataset, hyperspectral images, and challenges like the XOR problem and text analysis. In text analysis, CG-CNN employs an innovative embedding strategy that utilizes the context of neighboring words for classification, while in visual and signal data, it enhances feature extraction by exploiting spatial information. CG-CNN mimics the context-guided unsupervised learning mechanisms of biological neural networks and it can be trained to learn its features on limited-size datasets. Our experimental results on natural images reveal that CG-CNN outperforms comparable first-layer features of well-known deep networks such as AlexNet, ResNet, and GoogLeNet in terms of transferability and classification accuracy. In text analysis, CG-CNN learns word embeddings that outperform traditional models like Word2Vec in tasks such as the 20 Newsgroups text classification. Furthermore, ongoing development involves training CG-CNN on outputs from another CG-CNN to explore multi-layered architectures, aiming to construct more complex and descriptive features. This scalability and adaptability to various data types underscore the potential of CG-CNN to handle a wide range of applications, making it a promising architecture for tackling diverse data representation challenges.

INDEX TERMS Deep Learning, Contextual Guidance, Unsupervised Learning, Transfer Learning, Feature Extraction, Pluripotency.

I. INTRODUCTION

Deep learning approach has led to great excitement and success in AI in recent years Alzubaidi et al. (2021); Fisher et al. (2023); Goodfellow et al. (2016); LeCun et al. (2015); Ravi et al. (2017); Zhao et al. (2019). With the advances in computing power, the availability of manually labeled large data sets, and a number of incremental technical improvements, deep learning has become an important tool for machine learning involving big data Gao et al. (2020); Goodfellow et al. (2016); Hu et al. (2023); Zhao et al. (2019). Deep Convolutional Neural Networks (CNNs), organized in series of layers of computational units, use local-to-global pyramidal architecture to extract progressively more sophisticated features in the higher layers based on the features extracted in the lower ones Goodfellow et al. (2016); Zhao et al. (2019). Such incrementally built-up features underlie the remarkable performance capabilities of deep CNNs.

When deep CNNs are trained on gigantic datasets to classify millions of data into thousands of classes,

the features extracted by the intermediate hidden layers – as opposed to either the raw input variables or the task-specific complex features of the highest layers – come to represent efficiently the objective content of the data Bengio (2012); Gao et al. (2020); Iman et al. (2023); Mukhopadhyay et al. (2023); Zhu et al. (2023). Such objectively significant and thus inferentially powerful features can be used not only in the classification task for which they were developed, but in other similar classification tasks as well. In fact, having such features can reduce complexity of learning new pattern recognition tasks Favorov & Ryder (2004). Indeed, taking advantage of this in the process known as transfer learning Chen et al. (2023); Gao et al. (2020); Goyal & Sharma (2023); Iman et al. (2023); Poyatos et al. (2023); Salehi et al. (2023); Tang & Xie (2023); Unver & Sener (2023); Yosinski et al. (2014); Zhu et al. (2023) helps extract broad-purpose features to preprocess the raw input data and boost the efficiency and accuracy of special-purpose machine learning classifiers trained on smaller datasets Bengio (2012); Goodfellow et al. (2016); Priya & Padmapriya (2023); Zhao et al. (2019). Transfer learning is accomplished by first training a "base" broad-purpose network on a big-data task and then transferring the learned features/weights to a special-purpose "target" network trained on new classes of a generally smaller "target" dataset Yosinski et al. (2014).

Learning generalizable representations is improved with data augmentation, by incorporating variations of the training set datasets using predefined transformations Shorten & Khoshgoftaar (2019). Feature invariances, which have long been known to be important for regularization Iman et al. (2023); Zhu et al. (2023), can also be promoted by such means as mutual information maximization Favorov & Ryder (2004); Hjelm et al. (2019), or by penalizing the derivative of the output with respect to the magnitude of the transformations to incorporate a priori information about the invariances Simard et al. (1992), or by creating auxiliary tasks for unlabeled data Dosovitskiy et al. (2014); Ghaderi & Athitsos (2016).

Although supervised deep CNNs are good at extracting pluripotent inferentially powerful transferable features, they require big labeled datasets with detailed external training supervision. Also, the backpropagation of the error all the way down to early layers can be problematic as the error signal weakens (a phenomenon known as the gradient vanishing Arjovsky & Bottou (2017)). To avoid these difficulties, in this paper we describe a self-supervised approach for learning pluripotent transferable features in a single CNN layer without reliance on feedback from higher layers and without a need for big labeled datasets. We demonstrate the use of this approach on two examples of a single CNN layer trained first on natural RGB image datasets and then on hyperspectral data images. Of course, there is a limit to sophistication of features that can be developed on raw input patterns by a single CNN layer. However, more complex and descriptive pluripotent features can be built by stacking multiple CNN layers, each layer developed in its turn by using our proposed approach on the outputs of the preceding layer(s).

Self-supervised learning enables models to learn rich representations from unlabeled data, particularly in scenarios where labeled data are scarce or costly to obtain. This method operates by creating a pretext task, such as predicting part of the data from other parts, to learn general features that can be beneficial for downstream tasks. The efficacy of self-supervised learning hinges significantly on the contextual information, the situational data that surrounds and adds meaning to the primary data features. This contextual guidance helps in learning features that are more useful for the arbitrary down-stream prediction tasks on the data. Creating and learning to discriminate self-supervised contextual meta-classes is an exhaustive task to be performed in one pass and transfer learning helps self-supervision. Transfer learning allows a model trained to perform well for one prediction problem (source domain) to serve as a starting point for a related but different problem (target domain). By integrating self-supervised learning with contextual guidance and transfer learning, the proposed methodology allows the large set of contextual classes to be broken into smaller source domain tasks that can be iteratively used to train features across varied tasks.

This paper expands on our previous work Kursun et al. (2022); Kursun & Favorov (2019); Kursun et al. (2023) by enhancing and demonstrating the robustness and applicability of the Contextually Guided Convolutional Neural Networks (CG-CNN). The new contributions include:

- The contextually guided model has been extended to handle datasets from different modalities, further showcasing its versatility as a generic feature extraction and transfer learning approach.
- New experiments on various datasets, includ-

ing text classification with the 20 Newsgroups dataset, and the tactile VibTac-12 dataset have been added. These additions demonstrate CG-CNN's ability to adapt to both structured and unstructured data across different modalities.

- We have introduced a new methodology for estimating the usefulness of CNN features, advancing the theoretical framework of our approach and offering a new metric, "transfer utility," which quantifies the effectiveness of the CG-CNN in various self-supervised learning settings.
- We have added a demonstration of the model's performance on the classical XOR problem. This provides deeper insights into the model's capability to develop features with high pluripotency and transfer utility.

In Section 2, we briefly review transfer learning and neurocomputational antecedents of our unsupervised feature extraction approach. In Section 3, we present the proposed Contextually Guided CNN (CG-CNN) method and a measure of its transfer utility. We present experimental results in Section 4 and conclude the paper in Section 5.

II. BACKGROUND ON TRANSFERABLE FEATURE EXTRACTION

Deep CNNs apply successive layers of convolution (Conv) operations, each of which is followed by nonlinear functions such as the sigmoidal or ReLU activation functions and/or max-pooling. These successive nonlinear transformations help the network extract gradually more nonlinear and more inferential features. Besides their extraordinary classification accuracy on very large datasets, the deep learning approaches have received attention due to the fact that the features extracted in their first layers have properties similar to those extracted by real neurons in the primary visual cortex (V1). Discovering features with these types of receptive fields are now expected to the degree that obtaining anything else causes suspicion of poorly chosen hyperparameters or a software bug Yosinski et al. (2014).

Pluripotent features developed in deep CNN layers on large datasets can be used in new classification tasks to preprocess the raw input data to boost the accuracy of the machine learning classifier Gao et al. (2020); Zhu et al. (2023). That is, a base network is first trained on a "base" task (typically with a big dataset), then the learned features/weights are transferred to a second network to be utilized for learning to classify a "target" dataset Yosinski et al. (2014). The learning task of the target network can be a new classification problem with different classes. The base network's pluripotent features will be most useful when the target task does not come with a large training dataset. When the target dataset is significantly smaller than the base dataset, transfer learning serves as a powerful tool for learning to generalize without overfitting.

The transferred layers/weights can be updated by the second network (starting from a good initial configuration supplied by the base network) to reach more discriminatory features for the target task; or the transferred features may be kept fixed (transferred weights can be frozen) and used as a form of preprocessing. The transfer is expected to be most advantageous when the features transferred are pluripotent ones; in other words, suitable for both base and target tasks. The target network will have a new output layer for learning the new classification task with the new class labels; this final output layer typically uses softmax to choose the class with the highest posterior probability.

Biological neural networks like cortical areas making up the sensory cortex (similar to deep CNNs) are organized in a modular and hierarchical architecture Hawkins et al. (2017). Column-shaped modules (referred to as columns) making up a cortical area work in parallel performing information processing that resembles a convolutional block (convolution, rectification, and pooling) of a deep CNN. Each column of a higher-level cortical area builds its more complex features using as input the features of a local set of columns in the lower-level cortical area. Thus, as we go into higher areas these features become increasingly more global and nonlinear, and thus more descriptive Favorov & Kursun (2011); Favorov & Ryder (2004); Priya & Padmapriya (2023).

Unlike deep CNNs, cortical areas do not rely on error backpropagation for learning what features should be extracted by their neurons. Instead, cortical areas rely on some local guiding information in optimizing their feature selection. While local, such guiding information nevertheless promotes feature selection that enables insightful perception and successful behavior. The prevailing consensus in theoretical neuroscience is that such local guidance comes from the spatial and temporal context in which selected features occur Favorov & Ryder (2004). The reason why contextually selected features turn out to be behaviorally useful is because they are chosen for being predictably related to other such features extracted from non-overlapping

sensory inputs and this means that they capture the orderly causal dependencies in the outside world origins of these features Favorov & Ryder (2004); Kursun & Favorov (2024); Phillips et al. (1995).

III. CONTEXTUALLY GUIDED CONVOLUTIONAL NEURAL NETWORK (CG-CNN)

A. BASIC DESIGN

In this paper we apply the cortical context-guided strategy of developing pluripotent features in individual cortical areas to individual CNN layers. To explain our approach, suppose we want to develop pluripotent features in a particular CNN layer (performing convolution + ReLU + pooling) on a given data repository. We set up a three-layer training system (Figure 1) as:

1) The Input layer, which might correspond to a 2-dimensional field of raw pixels (i.e., a 3D tensor with two axes specifying row and column and one axis for the color channels) or the 3D tensor that was outputed by the preceding CNN layer with already developed features;
2) The CNN layer ("Feature Generator"), whose features we aim to develop;
3) The Classifier layer, a set of linear units fully connected with the output units of the CNN layer, each unit (with softmax activation) representing one of the training classes in the input patterns.

As in standard CNNs, during this network's training the classification errors will be backpropagated and used to adjust connection weights in the Classifier layer and the CNN layer. List of the symbols used in the rest of this section are summarized in Table 1.

While eventually (after its training) this CNN layer might be used as a part of a deep CNN to discriminate some particular application-specific classes of input patterns, during the training period the class labels will have to be assigned to the training input patterns internally; i.e., without any outside supervision. Adopting the cortical contextual guidance strategy, we can create a training class by picking at random a set of neighboring window patches (Figure 1). Being close together or even overlapping, such patches will have a high chance of showing the same object and those that do will share something in common (i.e., the same context). Other randomly chosen locations in the dataset – giving us other training classes – will likely come from other objects and at those locations the neighboring window patches will have some other contexts to share. We can thus create a training dataset $\mathbf{X} = \{x^t \mid 1 \leq t \leq CN\}$ of $C \times N$ class-labeled input patterns by treating C sets of N neighboring window patches – each set drawn from a different randomly picked data location – as belonging to C training classes, uniquely labeled from 1 to C. These inputs are small $a \times a \times b$ tensors, $a \times a$ patches (feature-maps) with b features. We will refer to each such class of neighboring data patches as a "contextual group."

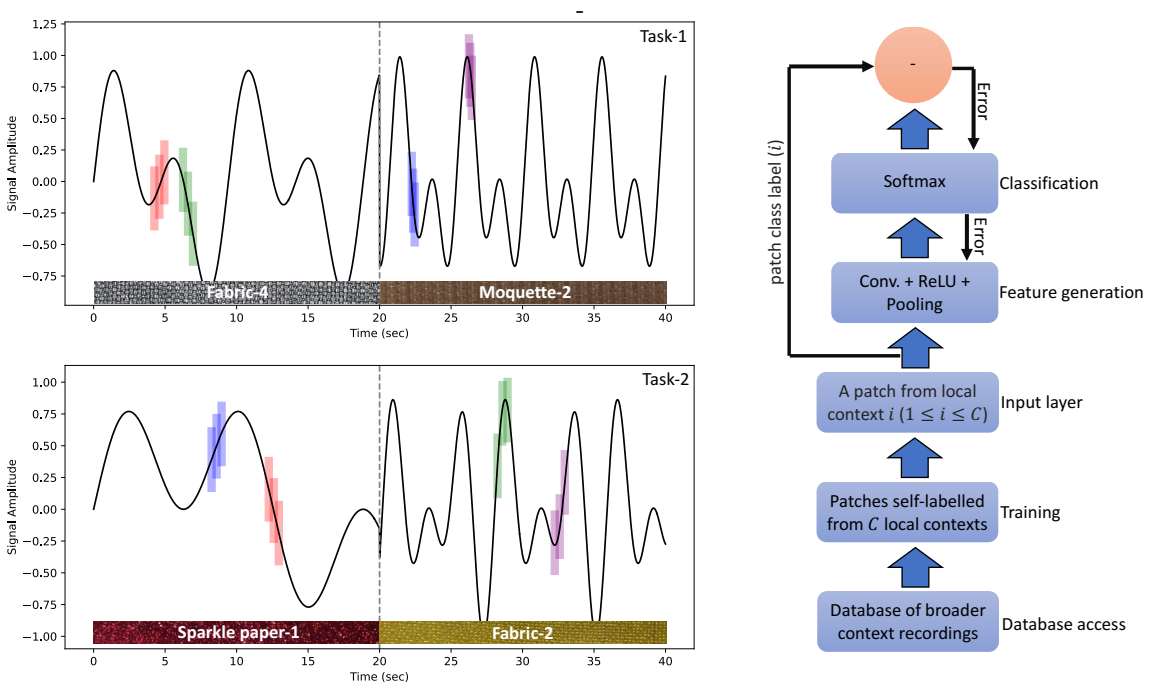
TABLE 1: List of Symbols

Symbol	Description
a	the width of the input bitmap data patches
$A_{CG}(C)$	transferable classification accuracy
b	the number of channels (e.g., red-green-blue) of the input bitmap data patches
C	the number of contextual groups
d	the number of feature maps (the features extracted from the $a \times a \times b$ input tensor
g	the extent of the spatial translation within contextual groups
N	the number of randomly chosen data patches in each contextual group
r^t	one-hot vector for the class label of x^t, where $r^t_c = 1$ if and only if $x^t \in$ contextual group c
s	the stride of the convolutions
U	Transfer Utility, estimates the pluripotency of the learned convolutional features
V_l	softmax weights of Layer-5 for contextual group $l \in \{1, 2, \ldots, C\}$ for the current task of discriminating C groups (each softmax unit has d input connections coming from y)
w	the kernel size of the convolutions
W_j	convolutional weights of unit j in Layer-2 (each unit/feature has $w \times w \times b$ input connections)
x^t	data patch number t used as input to CG-CNN
$\mathbf{X_E}$	the training dataset formed by x^t data patches ($1 \leq t \leq C \times N$) from C contextual groups with N patches in each group
$\mathbf{X_M}$	the dataset formed similarly as $\mathbf{X_E}$, which is used after the E-step for estimating the goodness-of-fit, A_{CG}, of the current Layer-2 weights W. It is also used for updating the Layer-2 weights (W_{new}) in the M-step
y^t	the d-dimensional feature vector computed as the output of the CNN layer
y^t_j	the response of unit/feature j in the CNN layer with $j \in \{1, 2, \ldots, d\}$
z^t_l	the response of output unit l in the Classifier layer to the input pattern x^t (estimate of the probability that x^t belongs to contextual group l)

Upon a presentation of a particular input pattern x^t from the training dataset $\mathbf{X}$, the response of the CNN layer is computed as:

$$y^t_j = MaxPool([W_j * x^t]^+) \quad (1)$$

where y^t_j is the response of output unit j in the CNN layer with d units (i.e., y_j is CNN's feature j, where $1 \leq j \leq d$), W_j is the input connection weights of that unit (each unit has $w \times w \times b$



(a) (b)

FIGURE 1: **(a)** Class-defining contextual groups of data patches. Each patch is shown as a small rectangular box superimposed on one of the database of recordings. Neighboring patches constitute a contextual group and during network training are treated as belonging to the same class. During network training, locations of contextual groups are picked at random. Two tasks and four groups per task are shown on this photo with three patches in each ($C = 4$ and $N = 3$). **(b)** CG-CNN architecture.

input connections), symbol * denotes convolution operation, and $[.]^+ = \max\{., 0\}$ denotes the ReLU operation. Next, the response of the Classifier layer is computed by the softmax operation as:

$$z_l^t = \frac{\exp\left(V_l \cdot y^t\right)}{\sum_{c=1}^{C} \exp\left(V_c \cdot y^t\right)} \tag{2}$$

where z_l^t is the response of output unit l in the Classifier layer (expressing the probability of this input pattern x^t belonging to class l), $y^t = [y_j^t]_{j=1}^d$ is the d-dimensional feature vector computed as the output of the CNN layer, and V_l is the vector of connection weights of that unit from all the d units of the CNN layer.

During training, connection weights W and V are adjusted by error backpropagation so as to maximize the log-likelihood (whose negative is the loss function):

$$\mathcal{L}(V, W|\mathbf{X}) = \sum_{t=1}^{CN} \sum_{c=1}^{C} r_c^t \log z_c^t \tag{3}$$

where $r_c^t \in 0, 1$ indicates whether input pattern x^t belongs to class c.

Table 1, listed in alphabetical order, provides the nomenclature of symbols used in our description of the CG-CNN algorithm.

B. ITERATIVE TRAINING ALGORITHM

If we want to develop pluripotent features in the CNN layer that will capture underlying contextual regularities in the data collections, it might be necessary to create tens of thousands of contextual groups for the network's training Dosovitskiy et al. (2014); Ghaderi & Athitsos (2016). We can avoid the complexity of training the system simultaneously on so many classes by using an alternative approach, in which training is performed over multiple iterations, with each iteration using a different small sample of contextual groups as training classes Finn et al. (2017). That is, in each iteration a new small (e.g., $C = 100$) number of contextual groups is drawn from the database and the system is trained to discriminate them. Once this training is finished, a new small number of contextual groups is drawn and training continues in the next iteration on these new classes without resetting the already developed

CNN connection weights.

For such iterative training of the CG-CNN system, we use an expectation-maximization (EM) algorithm Alpaydin (2014); Do & Batzoglou (2008). The EM iterations alternate between performing an expectation (**E**) step and a maximization (**M**) step. At each EM iteration, we create a new training dataset $\mathbf{X} = \{x^t \mid 1 \leq t \leq CN\}$ of $C \times N$ self-class-labeled input patterns and randomly partition it into two subsets; one subset $\mathbf{X_E}$ to be used in the E-step, the other subset $\mathbf{X_M}$ to be used in the M-step. Next, we perform the E-step, which involves keeping W connection weights from the previous EM iteration (W_{old}), while training V connections of the Classifier layer on the newly created $\mathbf{X_E}$ subset so as to maximize its log-likelihood $\mathcal{L}$ (Eq. 3):

$$\text{E-step: } V_{new} = \arg\max_{V} \mathcal{L}\big(V, W_{old} \big| \mathbf{X_E}\big) \quad (4)$$

Next, we perform the M-step, which involves holding the newly optimized V connection weights fixed, while updating W connections of the CNN layer on the $\mathbf{X_M}$ subset so as to maximize log-likelihood $\mathcal{L}$ one more time:

$$\text{M-step: } W_{new} = \arg\max_{W} \mathcal{L}\big(V_{new}, W \big| \mathbf{X_M}\big) \quad (5)$$

Overall, EM training iterations help CG-CNN take advantage of transfer learning and make it possible to learn pluripotent features using a small number of classes in the Classifier layer (each softmax unit in this layer represents one class). By continuing to update the CNN layer weights W, while the contextual groups to be discriminated by the Classifier keep changing with every EM iteration, CG-CNN spreads the potentially high number of contextual groups (classes) needed for learning data archive contextual regularities into multiple iterations Finn et al. (2017). The proposed EM algorithm for training CG-CNN achieves an efficient approach to learning the regularities that define contextual classes, which otherwise would theoretically require a C value in orders of tens of thousands Dosovitskiy et al. (2014).

To monitor the progress of CG-CNN training across EM iterations – so as to be able to decide when to stop it – we can at each EM iteration compute the network's current classification accuracy. Since we are interested in transferability of the CNN-layer features, such accuracy evaluation should be performed after the E-step, when Classifier-layer connections V have been optimized on the current iteration's task (using the $\mathbf{X_E}$ subset of input patterns), but before optimization of CNN-layer connections W (which were transferred from the previous EM iteration). Furthermore, classification accuracy should be tested on the new, $\mathbf{X_M}$, subset of input patterns. Such classification accuracy can be expressed as the fraction of correctly classified test ($\mathbf{X_M}$) input patterns. We will refer to such classification accuracy of CG-CNNs with task-specific Classifier weights V but transferred CNN feature weights W as "transferable classification accuracy" and use it in Section 4 as an indicator of the usefulness of context-guided CNN features on a new task:

$$A = \frac{1}{|\mathbf{X_M}|} \sum_{t=1}^{|\mathbf{X_M}|} \left[\arg\max_{c} r_c^t = \arg\max_{c} z_c^t\right] \quad (6)$$

where the argmax operators return the indices of the expected and predicted classes of x^t, respectively; and $[i = j]$ is the Kronecker delta function (expressed using the Iverson bracket notation) used to compare the expected and predicted classes.

As will be detailed further in Section 4, no particular CNN architecture is required for applying the CG-CNN training procedures. In Algorithm 1, we formulate the CG-CNN algorithm using a generic architecture (that somewhat resembles AlexNet because GoogLeNet, for example, does not use ReLU but it uses another layer called BatchNorm). Regardless of the particulars of the chosen architecture, CG-CNN accepts a small $a \times a \times b$ tensor as input. Although CG-CNN can be applied repeatedly to extract higher level features on top of the features extracted in the previous layer as mentioned at the beginning of this section, in this paper, focusing on CG-CNN's first application to data directly, b simply denotes the number of color bands, i.e. $b = 3$ for RGB image data, and a denotes the width of the resemblance patches that form the contextual groups. The kernel size of the convolutions and the stride are denoted by w and s, respectively. In CNNs, pooling operations, e.g. MaxPool, are used to subsample (that leads to the pyramidal architecture) the feature maps. A 75% reduction is typical, which is achieved via pooling with a kernel size of 3 and stride of 2, which gives us $a = w + 2s$. For example, if $w = 11$ and $s = 4$ for the convolutions (as in AlexNet), then $a = 19$. That is, CG-CNN's Feature Generator (the CNN layer) learns to extract d features (e.g., $d = 64$) that most contextually and pluripotently represent any given $a \times a$ data image patch. Note that at this level CG-CNN is not trying to solve an

Algorithm 1 The proposed CG-CNN method for learning broad-purpose transferable features

CG-CNN = [
  // the Input layer
  Layer-1: InputLayer(input_size = $a \times a \times b$)
  // the CNN layer (Feature Generator)
  Layer-2: ConvLayer(kernel_size = $w \times w \times b$, output_size = d, stride = s)
  Layer-3: ReLULayer
  Layer-4: y = MaxPool(kernel_size = 3×3)
  // the Classifier layer (Discriminator)
  Layer-5: z = Softmax(output_size = C)
]
Randomly initialize Layer-2 weights W
repeat// *Start a new EM iteration*
  // Create New Task
  // Populate a new dataset, $\mathbf{X}$ *with* C *classes and* N *instances per class*
  for c = 1 to C **do**
    Pick a random $a \times a$ window as the seed
    Spatial Contextual Guidance:
    Randomly slide the seed $\pm g$ pixels to create N instances of class-c.
    Color-based Contextual Guidance:
    Random 10% color-jitter to perturb brightness, contrast, saturation, and hue. 50% are converted to grayscale.
  end for
  Split $\mathbf{X}$ into the E-dataset and M-dataset
  // E-step:
  Set learning rate of Layer-2 to 0
  Randomly initialize Layer-5
  Train Layer-5 using E-dataset to get V_{new}
  Compute Accuracy, A, using V_{new} and W on M-dataset
  // M-step:
  Set learning rate of Layer-5 to 0
  Restore learning rate for W of Layer-2
  Fine-tune W by using the M-dataset
until A converges

actual classification problem and is only learning a powerful local representation; only a pyramidal combination of these powerful local features can be used to describe a data big enough to capture real-world object class.

C. PLURIPOTENCY ESTIMATION OF CNN FEATURES

EM training of CG-CNN aims to promote pluripotency of features learned by the CNN layer; i.e., their applicability to new classification tasks. Ideally, pluripotency of a given set of learned features would be measured by applying them to a comprehensive repertoire of potential classification tasks and comparing their performance with that of: (1) naïve CNN-Classifiers, whose CNN-layer connection weights are randomly assigned (once randomly assigned for a given task, these W weights are never updated as in extreme learning machines Wang et al. (2022); Glorot & Bengio (2010) present a state-of-the-art weight initialization method); and (2) task-specific CNN-Classifiers, whose CNN-layer connections are specifically trained on each tested task. The more pluripotent the CG features, the greater their classification performance compared to that of random features and the closer they come to the performance of task-specific features. Such a comprehensive comparison, however, is not practically possible. Instead, we can resort to estimating pluripotency on a more limited assortment of tasks, such as for example discriminating among newly created contextual groups (as was done in EM training iterations). Regardless of the C-parameter used in the CG-CNN training tasks, these test tasks will vary in their selection of contextual groups as well as the number (C) of groups.

The expected outcome is graphically illustrated in Figure 2, plotting expected classification accuracy of CNN-Classifiers with random, task-specific, and CG features as a function of the number of test classes. When the testing tasks have only one class, all three classifiers will have perfect accuracy. With increasing number of classes in a task, classification accuracy of random-feature classifiers will decline most rapidly, while that of task-specific classifiers will decline most slowly, although both will eventually converge to zero. CG-feature classifiers will be in-between. According to this plot, the benefit of using CG features is reflected in the area gained by the CG-feature classifiers in the plot over the baseline established by random-feature classifiers. Normalizing this area by the area gained by task-specific classifiers over the baseline, we get a measure of "Transfer Utility" of CG features:

$$U = \frac{\sum_{C=1}^{\infty} \mathbb{E}\big[A_{CG}(C)\big] - \sum_{C=1}^{\infty} \mathbb{E}\big[A_{random}(C)\big]}{\sum_{C=1}^{\infty} \mathbb{E}\big[A_{specific}(C)\big] - \sum_{C=1}^{\infty} \mathbb{E}\big[A_{random}(C)\big]} \tag{7}$$

where $A_{random}(C)$, $A_{specific}(C)$, and $A_{CG}(C)$ are classification accuracies of CNN-Classifiers with random, task-specific, and CG features, respectively, on tasks involving discrimination of C contextual groups (Eq. 6).

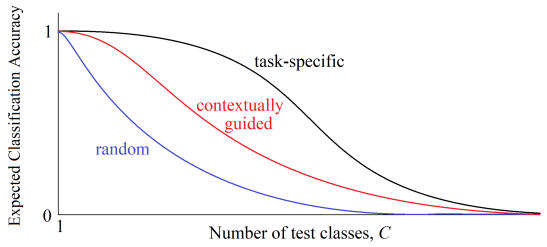


FIGURE 2: Transfer Utility of CG-CNN features is based on the area under the curve of the test accuracy A_{CG} as a function of the number of test classes C. Accuracies obtained using the random and task-specific CNN features, $A_{random}(C)$ and $A_{specific}(C)$ are also shown as they are used in Eq. 7 to quantify the Transfer Utility, U. The expectation of the test accuracies is computed over a number of tasks generated for each value of C.

D. SOURCES OF CONTEXTUAL GUIDANCE

While in our presentation of CG-CNN so far we have explained the use of contextual guidance on an example of spatial proximity, using neighboring data patches to define training/testing classes, any other kind of contextual relations can also be used as a source of guidance in developing CNN-layer features. Temporal context is one such rich source. Frequency domain context is another source, most obviously in speech recognition, while in Section 4 we exploit it in a form of hyperspectral imaging. More generally, any natural phenomena in which a core of indigenous causally influential factors are reflected redundantly in multivariable raw sensor data will have contextual regularities, which might be possible to use to guide feature learning in the CNN layer.

With regard to spatial-proximity-based contextual grouping, it is different from data augmentation used in deep learning Dosovitskiy et al. (2014). Data augmentation does shift input data patches a few pixels in each direction to create more examples of a known object category (such as a car or an animal); however, for a contextual group, we do not have such object categories to guide the placement of data patches and we take patches over a much larger range pixels from the center of the contextual group. Training CG-CNN using short shifts (similar to data-augmentation) does not lead to tuning to V1-like features because other/suboptimal features can also easily cluster heavily overlapping data patches.

Another source of contextual information that we utilize in this paper for extracting features from color data images is based on multiple pixel-color representations (specifically, RGB and grayscale). Instead of using a feature-engineering approach that learns to extract color features and grayscale features separately, as in Alzubaidi et al. (2021), we use a data-engineering approach by extending the contextual group formation to the color and gray versions of the data image windows: For every contextual group, some of the RGB data image patches are converted to grayscale. This helps our network develop both gray and color features as needed for maximal transfer utility: if the training is performed only on gray data images, even though the neurons might have access to separate RGB color channels, whose weights are randomly initialized and the visualization of feature weights initially looks colorful, they all gradually move towards gray features. Using no grayscale data images leads to all-color features automatically. For our experiments in Section 4, the probability for the random grayscale transformation was set to 0.5. That is, we converted 50% of the data image patches in each contextual group from color to gray, which led to emergence of gray-level features in addition to color ones.

IV. EXPERIMENTAL RESULTS

A. DEMONSTRATION ON THE XOR PROBLEM

The XOR problem stands as a fundamental example to demonstrate the importance of extracting nonlinear features and the effectiveness of multilayer networks. In addressing this problem, our method showcases its robust representational capacity, enhancing the variety of nonlinear mappings that can be captured. This capacity is akin to the principles seen in Radial Basis Function (RBF) networks or the concept of pluripotency (high Rademacher complexity) Favorov & Kursun (2011), emphasizing the preservation of nuanced neighborhood structures within the data. We utilize synthetic data to illustrate the efficacy of our approach, underscoring that its success is not reliant on convolutional processes or image data, which are common in CNN applications. Instead, this demonstration highlights how our methodology can discern patterns and learn relationships in data that are not inherently image-based, showcasing its adaptability and broader applicability beyond the conventional scope of image-focused neural networks. As outlined in Figure 3.a, this subsection presents the CG-CNN approach to solving the XOR problem through transfer learning, beginning with self-supervision on a task involving classification of random blobs, followed by fine-tuning for the XOR classification problem.

The initial phase of our experiment involved

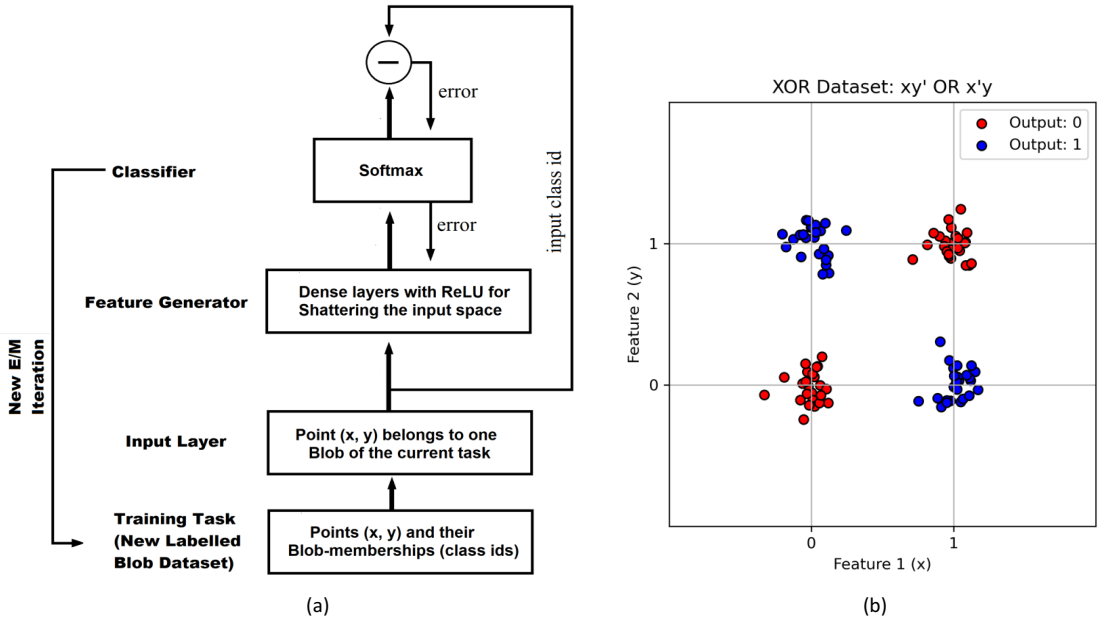


FIGURE 3: (a) Illustration of the CG-CNN architecture applied to the input space of the XOR dataset; (b) the XOR dataset represented using Gaussian blobs with a standard deviation of 0.1.

training a neural network in a self-supervised manner on a dataset comprising randomly generated blobs (shown in Figure 3.b). Each blob represented a distinct class, and the goal was to classify points into their respective blobs. This task was designed to encourage the model to learn rich feature representations that capture the underlying structure of the data space.

Our model architecture for this phase included two hidden layers with 20 neurons each, incorporating sigmoid activation functions. The output layer for this multi-class self-supervised task utilized a softmax activation function. Even though the number of output neurons (the number of classes in a task) changes, the hidden layers gradually learn features that can lead to better starting point than random initialization in order to obtain higher average classification accuracy (i.e. Transfer Utility) with shorter training on limited training data. We also integrated Dropout layers with a 0.1 dropout rate subsequent to each hidden layer in our model for regularization and to prevent over-reliance on particular features, thereby promoting generalization for all the models (even more so for the task-specific ones). By randomly deactivating a subset of neurons during training, Dropout ensures that different neurons are activated across iterations, reducing the likelihood of the model entirely losing previously learned information (catastrophic forgetting) from the previous self-supervised tasks. This approach helps preserve valuable features learned during the initial phase of training when the model is subsequently adapted to new tasks.

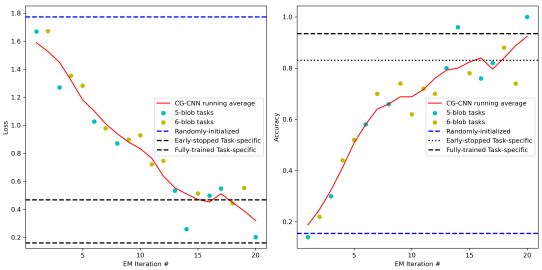


FIGURE 4: The transferable classification loss and accuracy are depicted in the left and right panels, respectively. As the network is exposed to more tasks, there is an evident emergence of a powerful representation, characterized by progressively lower loss and higher accuracy on self-supervised tasks that indicates a growing Transfer Utility. The smooth curve denotes the running average.

Upon completing the self-supervised learning phase, Figure 4 presents a sample run, the model develops nonlinear features of the input space that are capable of distinguishing between various blob regions (as shown and compared in Figures 5 and 6). We leveraged these learned features for the XOR

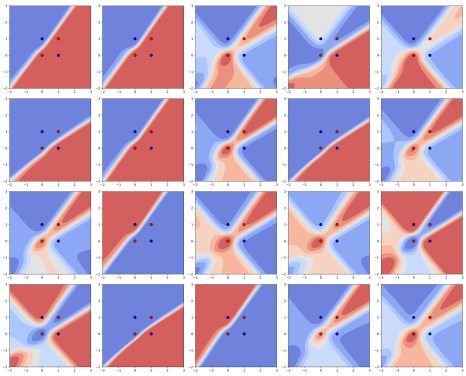


FIGURE 5: The task-specific features for the XOR dataset are tailored exclusively for solving the XOR problem and are not general-purpose. Unlike self-supervised learning methods, these features do not aim to develop versatile attributes applicable across a broader range of tasks.

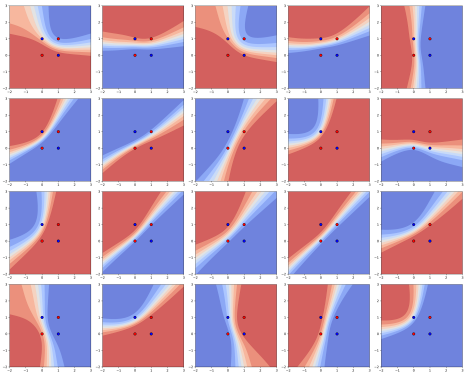


FIGURE 6: CG-CNN, trained on 20 diverse tasks (excluding the XOR task), learns to distinguish any patch of space from others and effectively partitions the input space more diversely. The features have the capacity to easily learn the XOR problem as well.

problem (Figure 3) as a canonical example of a non-linearly separable dataset. To adapt the model for the XOR problem, we modified the output layer to consist of a single neuron with a sigmoid activation function. Transferring the previously learned features, improved the model's ability to learn the XOR pattern. Our comparisons of the accuracy and loss of the CG-CNN based model with the random model and the task-specific model exemplifies the Figure 2. In Table 2, we present the average and standard deviation of accuracy and loss for 30 runs with 100 tasks and 4 to 7 blobs in each task.

B. DEMONSTRATION ON NATURAL IMAGES

To demonstrate the feasibility of CG-CNN in developing pluripotent features using a limited number of images without any class-labels, we employed images from the Caltech-101 dataset Bansal et al. (2023). Specifically, we utilized images from the face class to emphasize that the proposed algorithm operates without external supervision (having different classes such as human faces versus animals etc) for feature discrimination. This dataset comprised 435 color images with sizes around 400×600 pixels. We used half of these images to train CG-CNN and develop its features, and the other half to evaluate the pluripotency of these features. Adapting the description given in Figure 1, we applied the CG-CNN self-training by extracting smaller patches (e.g., 19×19 pixels) from these images as seeds and then creating a group from each sees by extracting nearby patches (shifting) and applying data augmentation techniques such as color-jitter and RGB-to-grayscale transformations. Each group was distinctively represented by a different class label, developed without relying on external supervision. The classes generated vary with each EM iteration, gradually enhancing the model's ability to find discriminatory features. Since our CG-CNN algorithm can be used with any CNN architecture, we applied it to AlexNet, ResNet, and GoogLeNet architectures. In its first convolutional block, AlexNet performs Conv+ReLU+MaxPool. This first block has $d = 64$ features, with a kernel size of 11×11 (i.e., $w = 11$) and a stride $s = 4$ pixels. ResNet performs Conv+BatchNorm+ReLU+MaxPool in its first block, with $d = 64$ features, kernel size of 7×7, and stride $s = 2$ pixels. GoogLeNet in its first block also has $d = 64$ features, 7×7 kernels, and stride $s = 2$. However, GoogLeNet performs Conv+BatchNorm+MaxPool. All three architectures use MaxPool with a kernel size of 3×3 and a stride $s = 2$. Therefore, the viewing window of a MaxPool unit is 19 pixels for AlexNet and 11 pixels for ResNet and GoogLeNet. (Note that although we could enrich these architectures by adding drop-out and/or local response normalization to adjust lateral inhibition, we chose not to do such optimizations in order to show that pluripotent features can develop solely under contextual guidance.)

We used a moderate number of contextual groups ($C = 100$) for the CG-CNN training. For selecting data patches for each contextual group, the parameter g – used in Algorithm 1 to slide the seed window for spatial contextual guidance – was set to $g = 25$ pixels. Thus, each contextual group had $(2 \times 25 + 1)^2 = 2601$ distinct patch positions. We

TABLE 2: Performance of models on the XOR Problem. Means and standard deviations over 30 runs are reported.

Model	Accuracy	Loss
CG-CNN Base with Top-Layer Trained Only	0.9793 ± 0.0555	0.3678 ± 0.0852
Randomly-Initialized Base with Top-Layer Trained Only	0.4987 ± 0.0503	0.6952 ± 0.0020
Task-Specific (Early-Stopped)	0.6447 ± 0.0856	0.6734 ± 0.0298
Task-Specific (Highly Trained)	0.9993 ± 0.0036	0.0011 ± 0.0028

FIGURE 7: Visualizations of the 11×11 weights of the 64 features in the CNN layer of CG-CNN after 1, 5, 20, 50, and 100 EM iterations. Also shown are the weights of the 64 features in the first layer of AlexNet. While even after 20 EM iterations the features are still quite crude, the features at iterations 50 and 100 are sharp and almost identical and resemble AlexNet features.

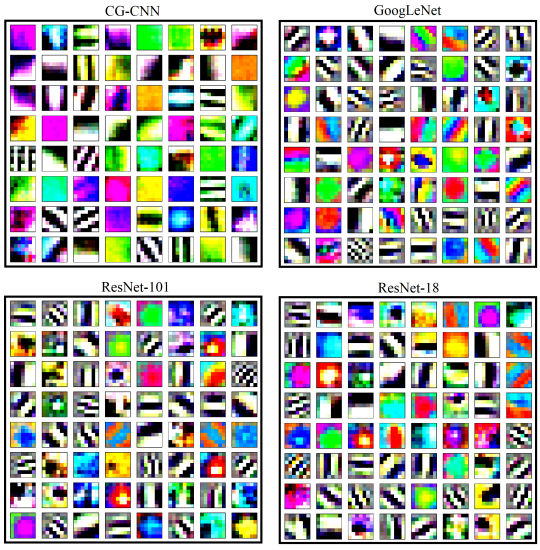


FIGURE 8: Visualizations of the 7×7 weights of the 64 features in the CNN layer of CG-CNN after 100 EM iterations, as well as 64 features in the first layer of GoogLeNet, ResNet-101, and ResNet-18.

also used color jitter and color-to-gray conversion to enrich contextual groups (see Section 3.4).

We used PyTorch open source machine learning framework Raschka et al. (2022) to implement CG-CNN. Experiments were performed on a workstation with Intel i7-9700K 3.6GHz CPU with 32 GB RAM and NVIDIA GeForce RTX 2080 GPU with 8GB GDDR6 memory. In each EM training iteration, we used 10 epochs for the E-step and 10 epochs for the M-step. On the workstation used for the experiments, for $C = 100$, each EM iteration takes about two minutes. CG-CNN takes around 100 minutes to converge in about 50 iterations. Both the SGD (stochastic gradient descent) and Adam Kingma & Ba (2017) optimizers can reduce time. Adam helps cut down the runtime by reducing the number of epochs down to one epoch with mini-batch updates. Increasing the number of EM iterations was more helpful than increasing the number of epochs in one iteration. With these improvements, 50 EM iterations took about 10 minutes, during which the Transferable Classification Accuracy, as formulated in Equation 6, initially starts around 30% and displays an upward trend throughout the training, specifically it rises quickly in the first few EM iterations and then slowly converges to a stable level around 60%. With each EM iteration, the network's features become progressively more defined and more resembling visual cortical features (gratings, Gabor-like features, and color blobs) as well as features extracted in the early layers of deep learning architectures AlexNet, GoogLeNet, and ResNet (see Figures 7 and 8).

To compare pluripotency of CG-CNN features to pluripotency of AlexNet, ResNet, and GoogLeNet features, we used the Transfer Utility approach described in Section 3.3 (see Figure 2 and Eq. 7) and tested classification accuracy of CNN-Classifiers equipped with random, task-specific, and CG features, as well as pretrained AlexNet, GoogLeNet, and ResNet features, on new contextual groups/-classes drawn from the test set that were not used during the training of CG-CNN. These classification accuracy estimates are plotted in Figure 9 as a function of the number of test classes C in each classification task. As the two plots in Figure 9 show, the curves generated with CG-CNN features lay slightly more than halfway between the curves

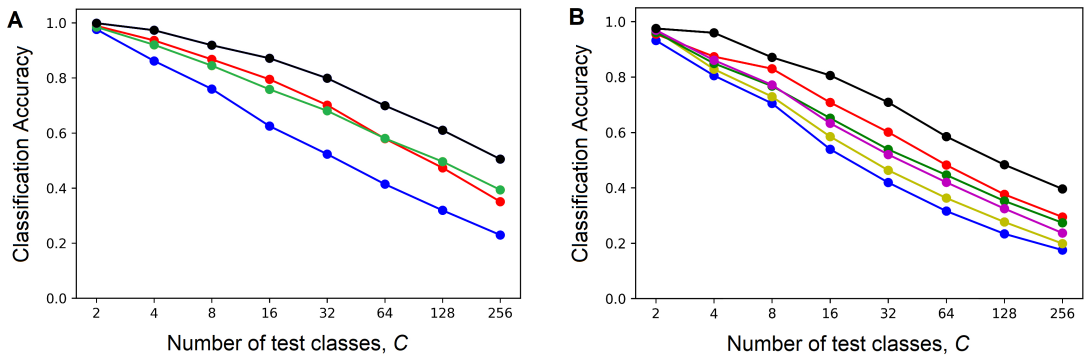


FIGURE 9: Transfer Utility of GC-CNN features, demonstrated following the format of Figure 2. **(A)** 11×11 pixel features. Average Classification Accuracies of CNN-Classifiers with task-specific, random, and CG features ($A_{specific}$, A_{random}, and A_{CG}; black, blue, and red curves, respectively) are plotted as a function of the number of contextual classes used in a classification task. For a comparison, also plotted is the average Classification Accuracy of CNN-Classifiers with Pool-1 features of the pretrained AlexNet (green curve). **(B)** 7×7 pixel features. Average Classification Accuracies of CNN-Classifiers with task-specific (black), random (blue), and CG (red) features are plotted as a function of the number of test classes. For a comparison, also plotted are the average Classification Accuracies of CNN-Classifiers with Pool-1 features of the pretrained GoogLeNet (green), ResNet-101 (magenta), and ResNet-18 (yellow) networks.

generated with random and task-specific features, indicating substantial degree of transfer utility. Most importantly, CG-CNN curves match or even exceed the curves generated with features taken from deep CNN systems, which are acknowledged – as reviewed in Sections 1 and 2.1 – to have desirable levels of transfer utility.

C. DEMONSTRATION ON TEXTURE CLASSIFICATION

As an additional test of pluripotency of CG-CNN features, we applied them to a texture classification task. Texture is a key element of human visual perception and texture classification is a challenging computer vision task, utilized in applications ranging from biomedical image analysis to industrial automation and remote sensing Anam & Rushdi (2019). For this test, we used the Brodatz dataset Brodatz (1966) of 13 texture data images, in which each data image shows a different natural texture and is 512×512 pixels in size (Figure 10). To compare with AlexNet (which has 11×11 pixel features, stride $s = 4$, and therefore pooled window size of 19×19 pixels), we trained classifiers to discriminate textures in 19×19 pixel Brodatz data patches. To compare with GoogLeNet and ResNet (which have 7×7 pixel features, stride $s = 2$, and therefore pooled window size of 11×11 pixels), we trained other classifiers to discriminate textures in 11×11 pixel Brodatz image data patches. For either of these two window sizes, we subdivided each 512×512 texture image data into 256 32×32 subregions and picked 128 training data patches at random positions within 128 of these subregions, and other 128 test data patches at random positions within the remaining 128 subregions. This selection process ensured that none of the training and test data patches overlapped to any degree, while sampling all the data territories.

Using the $128 \times 13 = 1664$ training data patches, we trained CNN-Classifiers equipped with either CG-CNN features (previously developed on Caltech-101 data images, as described above in Section 4.1), or AlexNet, GoogLeNet, or ResNet features. Note that these features were not updated during classifier training; i.e., they were transferred and used "as is" in this texture classification task. For additional benchmarking comparison, to gauge the difficulty of this texture classification task, we also applied some standard machine learning algorithms Fernandez-Delgado et al. (2014), including Decision Trees and Random Forests, Linear and RBF SVMs, Logistic Regression, Naive Bayes, MLP (Multi-Layer Perceptron), and K-NN (K-Nearest Neighbor).

These classifiers are straightforward to optimize without requiring many hyperparameters. For their implementation (including optimization/validation of the classifier hyperparameters), we used scikit-learn Python module for machine learning Raschka et al. (2022). For MLP, we used a single hidden layer with ReLU activation function (we used 64 hidden

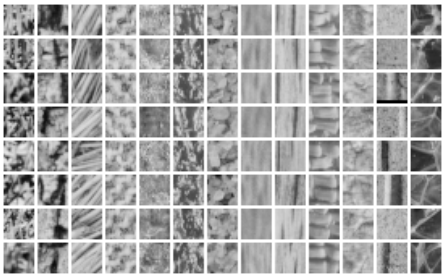


FIGURE 10: Brodatz texture data. Shown are 8 representative 19×19 pixel extracts from each of the 13 512×512 pixel images in the dataset.

units in the layer to keep the complexity similar to that of CG-CNN). We used the default value for the regularization parameter ($C = 1$) for our SVMs and the automatic scaling for setting the RBF radius for RBF-SVM. We used $K = 1$ neighbor and the Euclidean distance metric for K-NN. For the random forest classifier, we used 100 trees as the number of estimators in the forest.

All the classifiers were tested on the data patches from the test set, not used in classifier training. There are a total of $128 \times 13 = 1664$ test data patches. The accuracies of the classifiers are listed in Table 3. According to this table, all the CNN-classifiers with transferred features had very similar texture classification accuracies, with CG features giving the best performance. All the other classifiers performed much worse, indicating non-trivial nature of this classification task. These results demonstrate the superiority of using the transfer learning approach, with transferred features taken from CG-CNN or Pool-1 of pretrained deep networks.

D. DEMONSTRATION ON HYPERSPECTRAL IMAGES

Unlike color image processing that uses a large image window with a few color channels (grayscale or RGB), Hyperspectral Image (HSI) analysis typically aims at classification of a single pixel characterized by a high number of spectral channels (bands). Typically, HSI datasets are small, and application of supervised deep learning to such small datasets can result in overlearning, not yielding pluripotent task-transferrable HSI-domain features Sellami et al. (2019). To improve generalization, the supervised classification can benefit from unsupervised feature extraction of a small number of more complex/informative features than the raw data in the spectral channels. In this section, we demonstrate the usefulness of the features extracted by the

TABLE 3: Texture classification accuracies of 12 classifiers on 13 textures taken from the Brodatz (1966) dataset. Listed are means and standard deviations of the means computed over 10 test runs.

Method	11×11 **field**	19×19 **field**
CG-CNN	63.3 ± 0.7	74.3 ± 0.9
AlexNet		72.2 ± 0.7
GoogLeNet	62.2 ± 1.1	
ResNet-101	61.6 ± 0.9	
ResNet-18	61.5 ± 0.9	
RBF-SVM	53.6 ± 0.9	62.9 ± 0.9
Naive Bayes	39.3 ± 1.0	49.4 ± 0.8
Random Forest	34.7 ± 1.3	35.0 ± 1.5
MLP	33.6 ± 0.7	30.7 ± 0.6
K-NN	28.6 ± 0.9	29.2 ± 0.8
Linear-SVM	23.3 ± 1.2	28.0 ± 1.9
LR	22.8 ± 1.5	25.1 ± 0.6

proposed CG-CNN algorithm on two well-known HSI datasets Grana et al. (2018): (i) The Indian Pines dataset comprises a 145×145 pixel image data with 224 spectral bands in the 400-2500 nm range, including 16 classes such as crops, grass, and woods; (ii) The Salinas dataset features a 512×217 pixel HSI image data, also with 224 spectral bands, encompassing 16 classes like vegetables, bare soils, and vineyards.

To evaluate the quality of CG-CNN features on the hyperspectral data, the CG-CNN architecture presented in Algorithm 1 was used with the following parameters: $a = 3$ pixels, $b = 200$ bands for Indian Pines and 204 bands for Salinas (after discarding water absorption bands), $d = 30$ features, $w = 1$ pixel (i.e., each convolution uses only the bands of a single pixel), $C = 20$ contextual groups, $g = 2$ pixels for the extent of the spatial contextual guidance, and only 10 EM iterations. In training of CG-CNN, the class labels of the HSI pixels were not used; instead, local groups of pixels (controlled by the g parameter) were treated as training classes, as described in Section 3.4. CG-CNN learns to represent its input HSI data patch, which is a hypercube of size $3 \times 3 \times 200$, in such a way that the data patch and its neighboring windows/positions (obtained by shifting it $g = \pm 2$ pixels in each direction) can be maximally discriminable from other contextual groups centered elsewhere. Note that only a total of $(2 \times 2 + 1)^2 = 25$ data patches are created for each contextual group. We subsequently employed the extracted features as inputs for a Random Forest classifier. While the potential of multiple iterations of CG-CNN to produce even more descriptive and highly nonlinear features is acknowledged, this remains an avenue for future research due to its additional complexities. The primary focus of this study

TABLE 4: Comparison of RF Model Performance on the Indian Pines and Salinas Datasets.

Feature Type	Indian Pines	Salinas
Only Central Pixel with Original Bands (b Features)	86.3 ± 0.7	94.9 ± 0.1
Bands from 3×3 Neighborhood Merged ($9 \times b$ Features)	91.2 ± 0.3	95.1 ± 0.2
CG-CNN from 3×3 Neighborhood (30 Features Total)	96.2 ± 0.6	97.6 ± 0.2

is to showcase the improvement attained with these features compared to the raw ones (original bands). The use of Random Forest (RF) introduces ample nonlinearity Fernandez-Delgado et al. (2014), from which the original (raw) features can greatly benefit. However, it is observed that inputs derived from CG-CNN offer superior performance when used with RF, owing to their richer and more complex feature representation. This underscores the effectiveness of CG-CNN features in enhancing the RF classifier's performance as shown in Table 4. The results are averaged over 10 runs using a 70-30 split (excluding unlabeled pixels), with the default n_estimators=100 hyperparameter for the Random Forest classifier. CG-CNN's ability to learn robust embeddings through self-supervision not only enhances the accuracy of the downstream classifier but also enables faster training and testing times by reducing the number of features. This simplification is particularly beneficial for real-time applications.

E. DEMONSTRATION ON VIBTAC-12 VIBROTACTILE TEXTURE SIGNAL CLASSIFICATION

In this section, we study the performance of CG-CNN on a different dataset i.e., VibTac-12 Kursun & Patooghy (2020b) which is a dataset of vibrotactile signals collected from a 3-dimensional accelerometer sensor (MMA-7660 from NXP Company NXP Semiconductor (2012)). The signals are recorded as a probe scratches a rotating drum covered with various textures, simulating the sense of touch in humans. The diameter of the drum is $7\,\mathrm{cm}$ and it rotates at a linear speed of $5\,\mathrm{cm\,s^{-1}}$ which was chosen as a typical touch velocity. For each of the 12 textures, a 20-second recording is collected that corresponds to nearly five rotations of the drum. This dataset was collected and published in our previous work Kursun & Patooghy (2020a,b) using commercial off-the-shelf embedded boards and electrical components (AVR-based embedded boards, stepper motors, etc.) as well as our own designed and 3D printed mechanical components (including the rotating drum glued with different texture strips). The collected tactile dataset has 12 texture classes and Figure 11 shows an exemplary subset of texture strips that were used for the experiments. Textures include sandpapers of various grits, Velcro strips with various thicknesses, aluminum foil, and rubber bands of various stickiness.

In this study, we employ CG-CNN as a semi-supervised approach to tackle the classification challenges posed by the VibTac-12 dataset, integrating both supervised and unsupervised learning phases using the Expectation-Maximization (EM) methodology. The experiments use labeled examples from classes 1-10 and unlabeled examples from classes 11 and 12 to evaluate the transferability and effectiveness of the learned features across different sensory signals.

During the semi-supervised learning phase, the model leverages EM cycles to form meta-classes based on contextual similarities among data patches while simultaneously predicting class labels (when available). The process begins by selecting a seed data patch for each meta-class and expanding these meta-classes using additional input patches through data augmentation. An additional softmax layer attempts to predict the true class labels. In the E-step, two discriminators (softmax classifiers) are trained while keeping the feature extractor frozen, and in the M-step, the features are fine-tuned while the discriminators remain static. At the end of each M-step, a new EM iteration begins, creating a new task with a refreshed set of labeled and unlabeled examples.

Once the pretraining is completed, fine-tuning commences on a few labeled examples from classes 11 and 12, with the features frozen to evaluate their quality for the supervised learning task. It is important to note that self-supervised training of the CG-CNN was conducted on entirely distinct sets for self-supervision, training, and testing to prevent data leakage and ensure valid experimentation. Specifically, the fine-tuning process is executed on different segments of the signals that were not used during EM training. The model's classification accuracy is then evaluated on separate segments of classes 11 and 12.

Additional details of the experiments are as follows. We used 2-second recordings, corresponding to 400 timesteps per example, with 500 examples per class. For pretraining, we used data from 6 to 16 seconds, leaving 0.5 to 5.5 seconds for training on classes 11 and 12, and 16.5 to 19.5 seconds

FIGURE 11: Images of the 12 texture classes recorded in the VibTac-12 dataset Kursun & Patooghy (2020a)

for testing. The first and last 0.5 seconds were excluded to avoid noise. During pretraining, we defined 20 auxiliary classes and conducted 10 EM iterations, each consisting of 20 epochs for the E-step (training the softmax layers while keeping the features frozen) and 20 epochs for the M-step (fine-tuning the features). For the target task (classes 11 and 12), we used 50 labeled examples per class. We employed a 1D Convolutional Neural Network (CNN) for feature extraction, with the following architecture:

```
Conv1D(16, kernel_size=3, strides=2)
ReLU()
BatchNormalization()
MaxPooling1D(pool_size=2)
Conv1D(32, kernel_size=3, strides=2)
ReLU()
BatchNormalization()
MaxPooling1D(pool_size=2)
Conv1D(64, kernel_size=3, strides=1)
ReLU()
Dropout(0.2)
x = GlobalMaxPool1D()
supervised_output = Softmax(10)(x)
aux_output = Softmax(C)(x)
```

The model consists of three convolutional layers with 16, 32, and 64 filters, each followed by regularization layers such as batch normalization, pooling, and dropout. The network concludes with two softmax outputs: one for the supervised task and one for the auxiliary task. After pretraining with EM, we froze the learned features and fine-tuned the model on 50 labeled examples from classes 11 and 12. As described above, the pretraining, finetuning, and testing were done on separate segments to ensure no data leakage.

TABLE 5: Comparisons of methods on the target domain classification on the XYZ sensor-signal data. Results are represented as an average of 10 runs and their standard deviation.

Method	**Accuracy (%)**
Randomly Initialized CNN (trained only on the target domain with few labeled examples)	63.1 ± 8.3
CNN with Transfer Learning (features transferred from a labeled source domain)	65.0 ± 10.1
CG-CNN (trained using only contextual information from the source domain, fine-tuned on few labeled target-domain examples)	83.9 ± 11.4
Semi-supervised CG-CNN (trained using both labeled source data and unlabeled target-domain data, fine-tuned on few labeled target-domain examples)	93.9 ± 5.0

As shown in Table 5, the features learned by the semi-supervised CG-CNN significantly enhance performance in the downstream classification task with labeled data.

- **Randomly Initialized CNN**: Trained from scratch on the target domain using random initialization with only a few labeled examples. This approach establishes a baseline performance without leveraging any pre-learned features. It has 8,097 learnable parameters, including those in the binary classification output.
- **CNN with Transfer Learning**: This method transfers features learned from a labeled source domain but does not utilize contextual information. It only benefits from the abundant labeled examples in the source domain, leading to some improvements in classification accu-

racy on the target domain. During base training, it learns 8,682 parameters. The classifier layer is then replaced with a binary output instead of the original 10-output layer. The target model has 8,097 learnable parameters, benefiting from the initialization performed during the base training.

- **CG-CNN with Unsupervised Feature Transfer**: Trained using only contextual information from the source domain without utilizing class labels, followed by fine-tuning on a few labeled examples from the target domain. While this method demonstrates the effectiveness of unsupervised feature learning in improving target-domain performance, it does not take advantage of the available class labels in the source domain. It learns 9,332 parameters in the base network, where global pooling reduces the output to 64 final features. The softmax layer (with $C = 20$ output neurons) is then replaced with a single binary output layer, and the model fine-tunes 65 parameters of this new layer for the target task.
- **Semi-Supervised CG-CNN**: This approach combines both labeled source data and unlabeled target-domain data for training, followed by fine-tuning on a few labeled examples from the target domain. It achieves the highest accuracy by integrating semi-supervised learning with feature transfer. This method uses contextual information from both labeled and unlabeled examples and leverages class labels when they are available. It learns 9,982 parameters in the base network (including an extra softmax layer utilized for the supervised examples), where global pooling reduces the output to 64 final features. The top two softmax layers are then replaced with a single binary output layer, and the model fine-tunes 65 parameters of this new layer for the target task.

F. DEMONSTRATION ON WORD EMBEDDINGS AND TEXT CLASSIFICATION

We demonstrate the applicability of CG-CNN to word embeddings Mikolov et al. (2013), which transform the discrete, symbolic nature of language into a continuous, multi-dimensional space, enabling neural networks to effectively process text. To adapt CG-CNN for word embeddings, we utilized the 20 Newsgroups dataset, a collection of documents from various newsgroup discussions. In this example, our approach not only generates contextually-guided word embeddings from scratch but also finetunes pre-existing embeddings to better suit the corpus. To optimize the dataset for neural network training, we executed the following preprocessing steps. Tokenization, conducted using a tokenizer configured to retain the top 10,000 unique tokens based on frequency, ensured the model's focus on the most relevant and prevalent terms within the corpus. Stopwords were eliminated using the list of the nltk library because these commonly occurring words hold little to no discriminative value for our analysis. Furthermore, tokens representing numerical values and those comprising fewer than three characters were removed, as they often contribute negligible contextual information and can introduce additional noise into the data. To ensure model training with meaningful context, documents shorter than 100 tokens (words) were discarded. These preprocessing measures resulted in a refined corpus of 2,961 documents with 7,811 unique tokens. The zero performance, which refers to the accuracy of a naive classifier that always predicts the majority class, was 12%. The baseline performance when using Word2Vec involves averaging the 300-dimensional embeddings of the words in each document, followed by logistic regression (LR) as the classifier. Word2Vec+LR results in a mean accuracy of 60.89% with a standard deviation of 4.59%. Similarly, when Word2Vec feature averaging is followed by a random forest (RF) classifier, it yields a mean accuracy of 66.91% with a standard deviation of 4.15%. All results presented in this study are reported as averages over 10 independent runs, using different random train-test splits for each run to ensure the robustness and reliability of the findings.

For the self-supervised training of CG-CNN, we divided the entire dataset into three parts using a stratified split to preserve class priors: 50% for the self-supervised training set where embeddings and features are learned without labels, 25% for the supervised training set where these features are mapped to the Newsgroups classes, and the remaining 25% for the test set to evaluate the effectiveness of this mapping on other unseen labeled examples. The CG-CNN training treats each document of the self-supervised training set as a separate meta-class, with examples from each meta-class consisting of text blobs that were randomly selected within that document. However, CG-CNN uses only C classes in an EM iteration, but not all documents at the same time. In our experiments, we used $C = 20$. We used text blobs of 20 tokens in length for this purpose. We explored two methods for learning feature

TABLE 6: Configuration of a CG-CNN for fine-tuning Word2Vec's pre-existing word embeddings.

Layer	Parameters	Description
`Conv 1D`	filters=300, kernel_size=1, strides=1, input_shape=(20, 300), activation='relu'	300 features, processes tokens independently, no skipping of tokens, ReLU activation
`Global MaxPool 1D`	None	Outputs a single 300-dimensional vector representing the context
`Dense`	units=20, activation='softmax'	Classifies into 20 meta-classes using softmax

TABLE 7: Configuration of a CG-CNN designed for learning word embeddings from scratch.

Layer	Parameters	Description
`Embedding`	input_dim=7811, output_dim=300, input_length=20	Embeds 7811 tokens into 300D vectors, sequences of 20 tokens
`Global MaxPool 1D`	None	Outputs a single 300D vector representing the context
`Dense`	units=20, activation='softmax'	Classifies into 20 meta-classes using softmax

representations: fine-tuning an existing Word2Vec embeddings and training an embedding layer from scratch.

Fine-tuning Word2Vec: In this approach, the aim is to fine-tune the pre-trained embeddings for capturing patterns in the data not recognized by the pre-trained Word2Vec model. We utilized the sequence of 20 300-dimensional Word2Vec embeddings as input to a neural network comprising a convolutional layer followed by a ReLU activation and a global max pooling layer. This setup reduces each blob into a single 300-dimensional vector (summarizing the context), which is then classified into $C = 20$ randomly selected meta-classes using a softmax layer. We extracted 80 blobs (sequences of 20 tokens) for each meta-class, which were evenly divided into two sets of 40 for the E and M steps of the EM iterations, respectively. After training through these EM iterations, we input a single token into the network and captured the output from the ReLU layer, recording this as the newly fine-tuned 300-dimensional embedding for that token. The specifics of the model configuration are outlined in Table 6.

Training a New Embedding Layer: In contrast to the fine-tuning method, this experiment utilized a dedicated Embedding Layer instead of a Conv1D layer. This approach aims to construct embeddings from scratch by converting integer sequences (representing words) into dense vectors, which are then used to classify the context of tokens within documents. Table 7 illustrates the architecture of the model. In this model, the Embedding layer has a vocabulary size of 7811, with each word learned to be mapped to a 300-dimensional vector, and it accepts sequences of 20 tokens. GlobalMaxPooling1D is applied to reduce the spatial dimensions of the embedding output to the most significant features for each token sequence. The Dense layer then classifies these features into one of 20 meta-classes using a softmax activation function.

For both fine-tuning and learning from scratch, the CG-CNN model effectively learns to distinguish each document from all others. The features developed through this process are well-suited for document classification, a task that is relatively easier than distinguishing arbitrary text blobs from each other. Both random forest and logistic regression classifiers learn significantly better with CG-CNN features compared to using standard Word2Vec embeddings. This improvement is particularly pronounced when CG-CNN is used for fine-tuning (rather than learning an embedding from scratch with a small unlabelled dataset). For our comparisons, once trained, CG-CNN's embeddings are utilized similarly to those from Word2Vec. For each document, the embeddings are averaged, and this average is then used as input to classifiers such as logistic regression (LR) and random forest (RF). The performance metrics outlined below underscore the effectiveness of CG-CNN. Using the LR classifier on CG-CNN's fine-tuned embeddings achieved a mean accuracy of 75.36% with a standard deviation of 4.26%, demonstrating a significant improvement over the 60.89% baseline of Word2Vec. Additionally, a CG-CNN configuration extracting 50 dimensions (instead of 300) yielded a mean LR accuracy of 71.19%. This configuration was used as a robustness check, effectively serving to validate the model's performance and to ensure there were no anomalies or bugs affecting the results. Using the RF classifier on CG-CNN's fine-tuned embeddings achieved a mean accuracy of 70.76% with a standard deviation of 4.06%, demonstrating a significant improvement over the 66.91% baseline of Word2Vec. Additionally, using the LR classifier on CG-CNN's from-scratch embeddings achieved a mean accuracy of 74.08% with a standard deviation

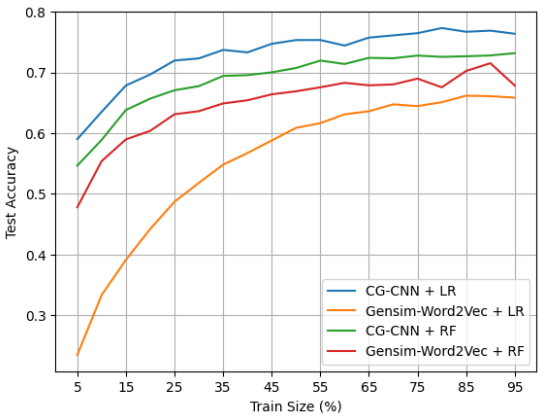


FIGURE 12: 20 Newsgroups test accuracy vs. train size using Logistic Regression (a) and Random Forest (b) on the original and fine-tuned embeddings.

of 1.06% and the RF classifier achieved a mean accuracy of 71.16% with a standard deviation of 1.13%.

In a departure from the previously reported 50-25-25 split for the unsupervised training, supervised training, and testing sets, we now vary the proportions of the training and testing sets within the labeled data. Figure 12 illustrates the average of 10 runs for the classification accuracies of logistic regression (LR) and random forest (RF) classifiers using CG-CNN fine-tuning versus Word2Vec embeddings. This is shown as a function of the train-size proportion, demonstrating how variations in training data proportions affect model performance.

V. CONCLUSIONS

The Contextually Guided Convolutional Neural Network (CG-CNN) presents a robust alternative to traditional deep learning models by emphasizing the development of pluripotent features through contextual guidance and transfer learning. This paper builds on our previous work by showcasing CG-CNN's broad applicability across various domains. Our experimental results, including applications to word embeddings for text classification with the 20 Newsgroups dataset and tactile sensing with the VibTac-12 dataset, demonstrate the adaptability of CG-CNN to both structured and unstructured data. Through its EM iterations, CG-CNN features progressively develop greater transfer utility—a concept formulated and defined in this paper as the degree of usefulness when applied to new tasks. Detailed demonstrations on the classical XOR problem further provide deep insights into the model's capacity to develop features with high transfer utility. CG-CNN features have shown superior classification accuracy compared to those from well-known deep networks like AlexNet, ResNet, and GoogLeNet, particularly in image-related tasks. Moreover, when fine-tuned through self-supervision on an unlabeled dataset, these features significantly outperform Word2Vec in text classification tasks on a downstream labeled test set. Overall, CG-CNN features provide substantial advantages in environments with limited data and requiring minimal model complexity.

The current single-layer CG-CNN, while effective, is limited in the complexity of the features it can develop. Future enhancements might involve expanding to a multi-layer architecture, where each layer could be trained using local contextual guidance from the outputs of preceding layers. Harnessing the contextual information at higher levels within data remains a critical challenge for guiding the development of higher-level CNN layers and determining the contextual groups for their EM-based training. Additionally, integrating feedback guidance from higher to lower layers could significantly enhance the CG-CNN model. Addressing the issue of feature forgetting —particularly for older tasks— remains as an unsolved issue; mechanisms to prevent this forgetting are necessary to maintain model efficacy over time.

ACKNOWLEDGMENT

This work was supported, in part, by the National Science Foundation under grants No. 2003740 and 2302537. The development of the semi-supervised learning algorithm benefited from the conceptual work involved in preparing NSF Grant No. 2435093, which is set to begin on November 1, 2024.

REFERENCES

Alpaydin, E. (2014). Introduction to machine learning, third edition. The MIT Press, Cambridge.

Alzubaidi, L., Zhang, J., Humaidi, A. J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., Santamaría, J., Fadhel, M. A., Al-Amidie, M., & Farhan, L. (2021). Review of deep learning: concepts, cnn architectures, challenges, applications, future directions. Journal of big Data, 8:1–74.

Anam, A. M. & Rushdi, M. A. (2019). Classification of scaled texture patterns with transfer learning. Expert Systems with Applications, 120:448 – 460.

Arjovsky, M. & Bottou, L. (2017). Towards princi-

pled methods for training generative adversarial networks. In International Conference on Neural Information Processing Systems (NIPS) 2016 Workshop on Adversarial Training. In review for ICLR, volume 2016.

Bansal, M., Kumar, M., Sachdeva, M., & Mittal, A. (2023). Transfer learning for image classification using vgg19: Caltech-101 image data set. Journal of ambient intelligence and humanized computing, pages 1–12.

Bengio, Y. (2012). Deep learning of representations for unsupervised and transfer learning. In Proceedings of ICML Workshop on Unsupervised and Transfer Learning, pages 17–36.

Brodatz, P. (1966). Textures: A photographic album for artists and designers. Dover Pubns.

Chen, W., Su, L., Chen, X., & Huang, Z. (2023). Rock image classification using deep residual neural network with transfer learning. Frontiers in Earth Science, 10:1079447.

Do, C. B. & Batzoglou, S. (2008). What is the expectation maximization algorithm? Nature Biotechnology, 26(8):897–899.

Dosovitskiy, A., Springenberg, J. T., Riedmiller, M., & Brox, T. (2014). Discriminative unsupervised feature learning with convolutional neural networks. In Advances in Neural Information Processing Systems, pages 766–774.

Favorov, O. V. & Kursun, O. (2011). Neocortical layer 4 as a pluripotent function linearizer. Journal of Neurophysiology, 105(3):1342–1360.

Favorov, O. V. & Ryder, D. (2004). Sinbad: A neocortical mechanism for discovering environmental variables and regularities hidden in sensory input. Biological Cybernetics, 90(3):191–202.

Fernandez-Delgado, M., Cernadas, E., Barro, S., & Amorim, D. (2014). Do we need hundreds of classifiers to solve real world classification problems? Journal of Machine Learning Research, 15:3133–3181.

Finn, C., Abbeel, P., & Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks.

Fisher, O. J., Rady, A., El-Banna, A. A., Emaish, H. H., & Watson, N. J. (2023). Ai-assisted cotton grading: Active and semi-supervised learning to reduce the image-labelling burden. Sensors, 23(21):8671.

Gao, F., Yoon, H., Wu, T., & Chu, X. (2020). A feature transfer enabled multi-task deep learning model on medical imaging. Expert Systems with Applications, 143:112957.

Ghaderi, A. & Athitsos, V. (2016). Selective unsupervised feature learning with convolutional neural network (s-cnn). 2016 23rd International Conference on Pattern Recognition (ICPR).

Glorot, X. & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In Teh, Y. W. & Titterington, M., editors, Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, volume 9 of Proceedings of Machine Learning Research, pages 249–256, Chia Laguna Resort, Sardinia, Italy. JMLR Workshop and Conference Proceedings.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT Press, Cambridge.

Goyal, V. & Sharma, S. (2023). Texture classification for visual data using transfer learning. Multimedia Tools and Applications, 82(16):24841–24864.

Grana, M., Veganzons, M., & Ayerdi, B. (2018). Hyperspectral remote sensing scenes - grupo de inteligencia computacional (GIC). http://www.ehu.eus/ccwintco/index.php. (Accessed on 12/22/2018).

Hawkins, J., Ahmad, S., & Cui, Y. (2017). A theory of how columns in the neocortex enable learning the structure of the world. Frontiers in Neural Circuits, 11:81.

Hjelm, R. D., Fedorov, A., Lavoie-Marchildon, S., Grewal, K., Bachman, P., Trischler, A., & Bengio, Y. (2019). Learning deep representations by mutual information estimation and maximization.

Hu, Z., Lin, L., Lin, W., Xu, Y., Xia, X., Peng, Z., Sun, Z., & Wang, Z. (2023). Machine learning for tactile perception: Advancements, challenges, and opportunities. Advanced Intelligent Systems, page 2200371.

Iman, M., Arabnia, H. R., & Rasheed, K. (2023). A review of deep transfer learning and recent advancements. Technologies, 11(2):40.

Kingma, D. P. & Ba, J. (2017). Adam: A method for stochastic optimization.

Kursun, O., Dinc, S., & Favorov, O. V. (2022). Contextually guided convolutional neural networks for learning most transferable representations. In 2022 IEEE International Symposium on Multimedia (ISM), pages 210–213.

Kursun, O. & Favorov, O. V. (2019). Suitability of features of deep convolutional neural networks for modeling somatosensory information processing. In Pattern Recognition and Tracking XXX, volume 10995, pages 94 – 105. International Society for Optics and Photonics, SPIE.

Kursun, O. & Favorov, O. V. (2024). Sinbad

origins of contextually-guided feature learning: Self-supervision with local context for target detection. In SoutheastCon 2024, pages 16–21.

Kursun, O. & Patooghy, A. (2020a). An embedded system for collection and real-time classification of a tactile dataset. IEEE Access, 8:97462–73.

Kursun, O. & Patooghy, A. (2020b). Vibtac-12: Texture dataset collected by tactile sensors.

Kursun, O., Sarsekeyev, B., Hasanzadeh, M., Patooghy, A., & Favorov, O. V. (2023). Tactile sensing with contextually guided cnns: A semisupervised approach for texture classification. In 2023 Seventh IEEE International Conference on Robotic Computing (IRC), pages 25–30.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553):436.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. Advances in neural information processing systems, 26.

Mukhopadhyay, M., Dey, A., & Kahali, S. (2023). A deep-learning-based facial expression recognition method using textural features. Neural Computing and Applications, 35(9):6499–6514.

NXP Semiconductor (2012). 3-axis orientation/-motion detection sensor. Document Number: MMA7660FC.

Phillips, W., Kay, J., & Smyth, D. (1995). The discovery of structure by multi-stream networks of local processors with contextual guidance. Network: Computation in Neural Systems, 6(2):225–246.

Poyatos, J., Molina, D., Martinez, A. D., Del Ser, J., & Herrera, F. (2023). Evoprunedeeptl: An evolutionary pruning model for transfer learning based deep neural networks. Neural Networks, 158:59–82.

Priya, G. S. & Padmapriya, N. (2023). Pt-cnn: A non-linear lightweight texture image classifier. Neural Processing Letters, pages 1–25.

Raschka, S., Liu, Y. H., Mirjalili, V., & Dzhulgakov, D. (2022). Machine Learning with PyTorch and Scikit-Learn: Develop machine learning and deep learning models with Python. Packt Publishing Ltd.

Ravi, D., Wong, C., Deligianni, F., Berthelot, M., Andreu-Perez, J., Lo, B., & Yang, G.-Z. (2017). Deep learning for health informatics. IEEE Journal of Biomedical and Health Informatics, 21(1):4–21.

Salehi, A. W., Khan, S., Gupta, G., Alabduallah, B. I., Almjally, A., Alsolai, H., Siddiqui, T., & Mellit, A. (2023). A study of cnn and transfer learning in medical imaging: Advantages, challenges, future scope. Sustainability, 15(7):5930.

Sellami, A., Farah, M., Riadh Farah, I., & Solaiman, B. (2019). Hyperspectral imagery classification based on semi-supervised 3-d deep neural network and adaptive band selection. Expert Systems with Applications, 129:246 – 259.

Shorten, C. & Khoshgoftaar, T. (2019). A survey on image data augmentation for deep learning. Journal of Big Data, 6:1–48.

Simard, P., Victorri, B., LeCun, Y., & Denker, J. (1992). Tangent prop - a formalism for specifying selected invariances in an adaptive network. In Moody, J., Hanson, S., & Lippmann, R. P., editors, Advances in Neural Information Processing Systems, volume 4. Morgan-Kaufmann.

Tang, H. & Xie, Y. (2023). Deep transfer learning for connection defect identification in prefabricated structures. Structural Health Monitoring, 22(3):2128–2146.

Unver, H. O. & Sener, B. (2023). A novel transfer learning framework for chatter detection using convolutional neural networks. Journal of Intelligent Manufacturing, 34(3):1105–1124.

Wang, J., Lu, S., Wang, S.-H., & Zhang, Y.-D. (2022). A review on extreme learning machine. Multimedia Tools and Applications, 81(29):41611–41660.

Yosinski, J., Clune, J., Bengio, Y., & Lipson, H. (2014). How transferable are features in deep neural networks? In Advances in Neural Information Processing Systems, pages 3320–3328.

Zhao, Z., Zheng, P., Xu, S., & Wu, X. (2019). Object detection with deep learning: A review. IEEE Transactions on Neural Networks and Learning Systems, pages 1–21.

Zhu, Z., Lin, K., Jain, A. K., & Zhou, J. (2023). Transfer learning in deep reinforcement learning: A survey. IEEE Transactions on Pattern Analysis and Machine Intelligence.