IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 31 March 2023, accepted 10 May 2023, date of publication 17 May 2023, date of current version 24 May 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3277253

RESEARCH ARTICLE

A Novel Similarity Based Unsupervised Technique for Training Convolutional Filters

TUĞBA ERKOÇ AND MUSTAFA TANER ESKİL
Pattern Recognition and Machine Intelligence Laboratory, Işık University, Şile, 34983 Istanbul, Turkey
Corresponding author: Tuğba Erkoç (tugba.erkoc@isikun.edu.tr)

This work was supported by the Scientific and Technological Research Council of Turkey (TUBITAK) under Grant 118E293.

ABSTRACT Achieving satisfactory results with Convolutional Neural Networks (CNNs) depends on how effectively the filters are trained. Conventionally, an appropriate number of filters is carefully selected, the filters are initialized with a proper initialization method and trained with backpropagation over several epochs. This training scheme requires a large labeled dataset, which is costly and time-consuming to obtain. In this study, we propose an unsupervised approach that extracts convolutional filters from a given dataset in a self-organized manner by processing the training set only once without using backpropagation training. The proposed method allows for the extraction of filters from a given dataset in the absence of labels. In contrast to previous studies, we no longer need to select the best number of filters and a suitable filter weight initialization scheme. Applying this method to the MNIST, EMNIST-Digits, Kuzushiji-MNIST, and Fashion-MNIST datasets yields high test performances of 99.19%, 99.39%, 95.03%, and 90.11%, respectively, without applying backpropagation training or using any preprocessed and augmented data.

INDEX TERMS Convolutional neural networks, feature extraction, unsupervised learning.

I. INTRODUCTION

Convolutional Neural Networks (CNNs) have become a widely used method for image classification tasks in recent years owing to their remarkable performance. Many techniques related to both the architecture and supervised training of CNNs have been proposed to obtain impressive state-of-the-art performance. New initialization techniques, activation functions, hyperparameter optimization methods, layer designs, skip connections, and data preprocessing are only a few examples that aim to achieve better performance via supervised learning. In each of these approaches, convergence depends on carefully built and complex architectures that require a sophisticated initialization scheme and optimization of hyperparameters.

Modern artificial neural networks(ANN) are conventionally trained using gradient-based error backpropagation. Prior to backpropagation, the features were handcrafted by domain experts. Backpropagation [1] enables the hidden layers of ANNs to learn features automatically based on a labeled input training set. This development has enabled the eventual emergence and success of CNNs in image classification.

Research on CNNs concentrates on supervised learning by proposing mechanisms to avoid the vanishing gradient problem [2] of backpropagation training. Collecting and annotating datasets large enough for supervised training is costly and error-prone because it depends on the perception of the person who labels the data. Methods attempting to reduce the number of labeled examples using active learning for the training of CNNs have been proposed [3], [4]. However, an oracle (supervisor) is still required for label queries in such systems. Moreover, humans naturally observe and learn in an unsupervised manner as mentioned in [5] so CNNs should be able to mimic this because the more complicated the architecture the more labeled data they require for training.

One requirement of gradient-based approaches is suitable initialization of weights. The convergence of the stochastic search to achieve the best feature extractors depends on careful initialization of the filter weights. Simple Gaussian initialization was shown [6] to be a poor choice for the initialization of the weights. Thus, the initialization of the weights is commonly carried out by picking random values from a

The associate editor coordinating the review of this manuscript and approving it for publication was Jon Atli Benediktsson.

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License.
For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

specific distribution [6], [7] regardless of the input domain. Depending on how the network is initialized, a stochastic search may overlook the crucial visual cues in the training set.

Hyperparameter tuning is another crucial factor affecting the performance of CNNs. Several optimization techniques [8], [9], [10], [11], [12] have been proposed to configure hyperparameters. However, hyperparameter optimization is tedious and time consuming. The running time of the optimization increases exponentially as the number of hyperparameters increases. An unsupervised learning approach can decrease the number of hyperparameters that must be tuned, thereby reducing the search time.

Unsupervised learning methods that have been proposed in recent years revolve around generative networks, clustering, and self-supervised learning, in which a new unlabeled task is defined and solved for training CNNs via pseudo-label-based backpropagation. We follow a different path from the existing unsupervised approaches for training the filters in the convolutional layers.

In this study, we propose an unsupervised backpropagationless training approach that does not use any pretext tasks, pseudo-labels, or generative models to train the filters of the convolutional layers. Our method can extract features from a given dataset without backpropagation, and the extracted filters perform close to purely supervised methods when used in a CNN model. This unsupervised method trains one convolutional layer at a time. The training starts with an empty layer, which is gradually filled with the discovered filters. The filters are discovered simply by taking a single look at the training images using a filter discovery rule based on the similarities of the features in the input domain. The contributions of this study are as follows:

- We show that an unsupervised backpropagationless training algorithm can train convolutional layers by looking at training samples once without the need for any labels.
- Our approach allows weight initialization methods to be discarded because the filters are extracted heuristically from the input domain, and the weights of the filters are determined by our unsupervised training algorithm.
- With our approach, there is no need to use costly hyperparameter optimization methods to set the number of filters because the number of filters hyperparameter is determined during training by our algorithm.

We applied our unsupervised algorithm to the raw MNIST, EMNIST-Digits, Kuzushiji-MNIST, and Fashion-MNIST datasets. Our approach achieves 99.19%, 99.39%, 95.03%, and 90.11% accuracy on the MNIST, EMNIST-Digits, Kuzushiji-MNIST, and Fashion-MNIST datasets, respectively, without any preprocessing or data augmentation. The remainder of the paper is organized as follows: Section II recounts the related work; Section III discusses the methodology employed for the development of the proposed algorithm; Section IV discusses the datasets used in the study, implementation of experiments, and their results; Section V discusses related to the proposed work; and Section VI concludes the paper.

II. RELATED WORK

Although research on deep networks is dominated by supervised learning, research on unsupervised learning has also been conducted in this field. Unsupervised methods can be classified into self-supervised learning, cluster-based learning, and generative models.

In self-supervised learning, data labels are replaced with pseudo-labels by performing pretext tasks. The main idea is to exploit the visual data for labeling. Surrogate classes are generated [13] by applying transformations such as translation, scaling, rotation, and contrast manipulation to randomly sampled seed image patches containing object or object parts in them. Subsequently, the surrogate classes are labeled and a CNN is trained with backpropagation to discriminate between these surrogate classes. Relative positions of the image patches are used as a pretext task by [14] and [15], which cut the images into pieces and shuffles them to create a jigsaw from which the network learns to solve the puzzle. Image colorization is applied as a pretext task by [16], [17], and [18], whereas [19] learns to predict pixels based on the surrounding pixel information via image in-painting. Video frames are tracked as a pretext task for motion cues by [20] and [21]. Stereo images are combined [22] to obtain drivable space and surface normals data to generate pseudo-ground truths as a pretext task. Ranking is used as a pretext task to assess the image quality in [23].

Cluster-based unsupervised learning methods often involve k-means [24] or Gaussian Mixture Models (GMM). The aim is to generate clusters for pseudo-labeling the training examples and training the network with backpropagation using these pseudo-labels. Thus, the performance of the models depends on the clustering accuracy because pseudo-labels are used in backpropagation. Agglomerative clustering is applied to the outputs of a CNN [25], and both the CNN and clusters are updated jointly on each backward pass based on the cluster labels until a stopping condition is met. A similar joint update of clusters and CNN weights is proposed by [26]. In Deep Embedded Clustering (DEC) [27], stacked auto encoders (SAE) are used to map input images to the feature space. K-means clustering is then applied to the outputs of the deep neural network to initialize the cluster centroids. This is followed by applying Kullback-Leibler (KL) divergence to refine the clusters. In [28], clusters are generated using GMM after dimension reduction is applied to the spatial vector output from a randomly initialized CNN. The generated cluster assignments are then used to update the CNN. The features are extracted using the trained CNN. Instead of using all the pseudo-labels of clusters, [29] trains multiple Auto Encoders (AE) in parallel and only selects the agreed-upon cluster pseudo-labels for training. ResNet-50 [30] is trained with the ImageNet [31] dataset by [32]

using the self-supervised learning method proposed in [15] and [33]. This pretrained network is used to generate clusters from another dataset via k-means and assigns pseudo-labels to these clusters. As the last step, a new model with the same architecture as Resnet-50 is trained using the new dataset and pseudo-labels from scratch by minimizing cross-entropy.

Generative models include AEs and Generative Adversarial Networks (GAN). AEs learn the mapping between the input data and their lower dimensional latent representation by unsupervised training of the network based on reconstruction. Convolutional Auto Encoder (CAE) is proposed [34] to learn localized feature representations which are subsequently used for weight initialization in CNNs. Hou et al. [35] applies CAE to finger-vein verification. Stacked denoising AEs are shown [36] to learn Gabor-like edges and performs better in classification of the MNIST digits than ordinary stacked AEs. In contrast to reconstructing from all hidden units, k-sparse AE [37] selects the top-k hidden units and discards others for better classification accuracy. GANs [38] consist of two networks: a generative model and a discriminative (adversary) model. The generative model generates new images from a given dataset to fool the discriminative model, and the adversary attempts to distinguish between real and generated samples. The mutual information between the GAN's noise variables and observations is maximized [39] for learning meaningful representations without labels. DCGAN [40], a new generator-discriminator CNN architecture which is used as an unsupervised feature extractor while [41] learns features from synthetically generated images using GAN.

In summary, the unsupervised literature can be categorized into three main categories: self-supervised learning, cluster-based learning, and generative models. Even though we refer to some unsupervised research in the previous paragraphs, they all still use backpropagation in the training, as opposed to our proposal of no backpropagation training. Both self-supervised and cluster-based learning methods in the literature assign pseudo-labels to their training data using either pretext tasks or clustering methods. These pseudo-labels are used to train the models using backpropagation. There are two networks in generative models. The generative model attempts to generate new images that can be as close to the original training images as possible to fool the discriminative model. The discriminative model attempts to distinguish between generated and real data. Both generative and discriminative models use backpropagation during the training.

III. METHOD

The method we propose is pure unsupervised training of convolutional layers, inspired by Fukushima's Winner-Take-All(WTA) [42] and Add-if-Silent(AiS) [43] rules. The training of convolutional layers depends on the discovery of new features based on a measure of similarity between the discovered features and the observation.

The more similar an input is to a feature, the higher the similarity score. This may seem similar to how Siamese Networks learn the similarity function among image triplets [44]. Unlike Siamese networks, we work only with a single CNN model that calculates the similarity of a candidate feature to any of the discovered features using dot product. Based on this similarity score, our method declares this candidate as a new feature or supporter of an existing feature. If the similarity score of the candidate is lower than a predefined similarity threshold, a new feature is discovered; otherwise the most similar (i.e., the winner) feature's weights are updated in an unsupervised manner. This discovery scheme allows us to remove the number of filters hyperparameter from the CNN architecture.

Contrary to the conventional CNN approach, in which thousands of epochs are required for training, our method discovers the filters from the training set in a self-organizing manner by looking at each training sample only once. This allows us to completely ignore the filter initialization techniques and the subsequent training of the filters via backpropagation due to the random initialization of the weights.

In this section, we begin with a brief introduction to the Neocognitron architecture, WTA, and AiS rules. We continue with the details of our unsupervised approach.

A. NEOCOGNITRON

The first CNN architecture based on the findings of [45] is proposed by Fukushima [46] in 1980. In contrast to the most recent CNN implementations, Fukushima's Neocognitron is not trained using a gradient based algorithm. The architecture is composed of S and C cell planes, which are the predecessors of the convolutional and pooling layers in modern CNNs.

Early Neocognitron architecture training is shaped around the WTA approach. According to the WTA, when an input pattern is introduced to the system, simple cells compete with each other to be the winner. The simple cell with the highest response to the input becomes the winner of the cell plane that it belongs to. Simple cell planes eventually become selectively sensitive to specific features due to the self-organization provided by the WTA algorithm. In [42], the WTA rule is utilized with a similarity threshold to determine the generation of new filters.

Later, in 2013, Fukushima proposes another training rule for Neocognitron termed AiS [43]. According to the AiS algorithm, when none of the current simple cells are activated for an input stimulus, this stimulus is used to form a new simple cell. The values of the stimulus vector that generates a new cell are used as the weights of the new cell. The weights of this cell are fixed and never updated. Therefore, the AiS rule maps all features in the input stimuli as simple cells.

In our approach, we integrate the ideas of the WTA and AiS rules for the discovery of convolutional layer filters among filter candidates drawn from the training images. We update

the weights of the discovered filters in an unsupervised manner in contrast to the AiS rule.

B. CONVOLUTIONAL FILTER DISCOVERY

We employ an unsupervised two-step evaluation process for the filter extraction from a given dataset. The first step involves the extraction of filter candidates from the training dataset. The second step is the unsupervised selection of filters from the candidates based on a similarity threshold. The entire process examines the training images only once, in contrast to backpropagation, in which multiple passes of the training images are required to train the filters. After the proposed method examines each image in the training set once, the training of the filters in the current convolutional layer is considered complete and no additional weight updates are applied to the discovered filters. The first step is executed on the CPU, whereas convolution and max-pooling operations are executed on the GPU. The details of the implementation of these two steps are explained in the following sections.

1) CENTER OF GRAVITY BASED CANDIDATE FILTER EXTRACTION

Convolutional filters are feature detectors that operate on a receptive field. Based on this fact, we extract the filter candidates from the input training images by cropping patches with strides of 1, as in our previous study [47]. However, cutting out patches from training images in this fashion leads to a very high number of candidates that often include no useful information. The number of the candidates directly affects the filter extraction running time because the similarities are calculated using computationally intensive cross-correlation in [47]. Thus, we propose an elimination process to maintain the set of filter candidates at an acceptable size while preserving crucial information. This elimination process is composed of variance and center of gravity (CoG) based elimination procedures. In combination with a reduced candidate set, we also calculate the similarities using dot product instead of cross-correlation, which leads to a huge speed gain (up to 16 times) while improving the accuracy compared to our previous work [47].

Before the first elimination step, all possible image patches are cut out from the input images with a specified stride and shape. However, some patches are expected to carry no useful information consisting of only the background. Therefore, we discard image patches with a variance of 0.

The second elimination step is based on the CoG of the image patch. While cutting out the image patches, if the stride value is small, several image patches containing the same feature in different positions are obtained. To remove redundant image patches, we calculate the CoG for each image patch. If the calculated CoG value is not within the range of $\pm$ 0.5 pixels in both the x and y axes from the center of the image patch, it is discarded. The remaining image patches form a set of filter candidates. The eliminations applied to the image patches ensure that attention is on the features that are useful in extracting the key elements from the training images. The same filter candidate extraction process is applied to the feature maps in the deeper layers.

Algorithm 1 Unsupervised Filter Learning

Input: images/feature maps
Output: filter weights matrix $\mathbf{W^L}$

$f_0 \leftarrow$ **True**
if $\mathbf{L} \neq \mathbf{0}$ **then**
  Obtain feature maps
end if
for each image/feature map **do**
  Obtain candidate filters set $\mathbf{C}$
  if $\mathbf{L}$ is empty **then**
    Set weights of the new filter $\mathbf{W_0^L}$ with $\mathbf{C_0}$
    Set supporter count such that $\varsigma_0^L \leftarrow 1$
    Set number of filters such that $\eta^L \leftarrow 1$
    continue
  end if
  for each candidate $\mathbf{C_i}$ **do**
    Calculate similarity scores $\vec{S}$
    if highest similarity score $\vec{S}_j >$ threshold **then**
      Update weights of the similar filter $\mathbf{W_j^L}$ with (1)
      Increment supporter count such that $\varsigma_j^L \leftarrow \varsigma_j + 1$
    else
      Set weights of the new filter $\mathbf{W_n^L}$ with $\mathbf{C_i}$
      Set supporter count such that $\varsigma_n^L \leftarrow 1$
      Increment number of filters such that $\eta^L \leftarrow \eta^L + 1$
    end if
  end for
end for
for each $\mathbf{W_j^L}$ **do**
  Stretch filter weights $\mathbf{W_j^L}$ to $[-1, 1]$
  Update filter positive weights with (2)
  Update filter negative weights with (3)
end for
$\mathbf{W^L}.trainable \leftarrow$ **False**

2) EXTRACTION OF FILTERS FROM THE CANDIDATES SET

After obtaining the filter candidates set $\mathbf{C}$ from the input images (or feature maps in the hidden layers), the next step is the extraction of the filters among the candidates.

In the proposed method, the convolutional layers always start as a blank slate, and the filters are discovered dynamically from the data observed in $\mathbf{C}$. The training algorithm depends on the similarity between the candidate and current layer filters to discover new ones. We train the convolutional layers one at a time using Algorithm 1.

The first convolutional layer is the input layer, which receives the training images as inputs. Thus, unprocessed training images are used to obtain the filter candidates set $\mathbf{C}$. According to our method, each layer $\mathbf{L}$ is initially empty. Thus, the first candidate filter selected from set $\mathbf{C}$ becomes the first discovered filter.

Each filter has a supporter count, which is crucial for the weight updates. This signifies the number of times the associated filter becomes the winner (i.e., the most similar filter to the candidate with a similarity beyond the threshold value) when a candidate is presented to the layer. Its initial

value is set to 1 when the candidate is established as a new filter for the current layer.

After the first filter is discovered, the algorithm starts judging the remaining candidates one by one against the discovered filters to decide whether they carry features that are different enough to become new filters. To determine whether a candidate $\mathbf{C_i}$ can become a new filter, similarity scores $\vec{S}$ are calculated for the candidate $\mathbf{C_i}$ with all previously discovered filters $\mathbf{W^L}$ of the current layer $\mathbf{L}$. The similarities are calculated with dot product instead of cross-correlation which leads to a large speed gain with the reduced candidate set (up to 16 times) while improving the accuracy compared to [47]. Each discovered filter in layer $\mathbf{L}$ wants to represent the remaining candidates. To represent a candidate, the filter $\mathbf{W_j^L}$ must be the most similar filter to that candidate. Therefore, the filter with the highest similarity score $\vec{S}_j$ becomes the winner for candidate $\mathbf{C_i}$. However, being the most similar filter is not sufficient to represent the candidate. The similarity score $\vec{S}_j$ must also be above a predefined similarity threshold, which indicates that the candidate $\mathbf{C_i}$ carries a pattern that has already been encountered in the past. Thus, this candidate cannot become a new filter, but a supporter of an existing filter (i.e., the winner). In this case, our algorithm updates the weights of the winner filter $\mathbf{W_j^L}$ according to (1), and the supporter count of the winner filter is incremented by one.

$$\mathbf{W_j^L} \leftarrow \mathbf{W_j^L} + \frac{\mathbf{C_i} - \mathbf{W_j^L}}{\varsigma_j^L + 1} \tag{1}$$

If the similarity score $\vec{S}_j$ is less than the similarity threshold value, the algorithm determines that the candidate $\mathbf{C_i}$ carries a pattern that has not been previously encountered. Thus, candidate $\mathbf{C_i}$ is a new discovery for the current layer $\mathbf{L}$. The filter weights are set with weights of $\mathbf{C_i}$ and the supporter count becomes 1. This process is illustrated in Fig.1. The filter discovery scheme is repeated until all candidates in $\mathbf{C}$ are processed.

After all candidates are processed, we apply a normalization routine to the discovered filters as opposed to our previous work [47]. The normalization routine begins by stretching the filter weights to the [-1,1] interval. Then, the positive and negative filter weights are updated separately using (2) and (3). The last step of the proposed algorithm is to set the filter weights $\mathbf{W^L}$ as nontrainable, because we do not allow training these filters any further with backpropagation.

$$\mathbf{W_j^{L(+)}} \leftarrow \mathbf{W_j^{L(+)}} / |\sum \mathbf{W_j^{L(+)}}| \tag{2}$$

$$\mathbf{W_j^{L(-)}} \leftarrow \mathbf{W_j^{L(-)}} / |\sum \mathbf{W_j^{L(-)}}| \tag{3}$$

In order to utilize the output of our algorithm, we create a Keras [48] Sequential model and add a convolutional layer to this model. Keras expects hyperparameter values for the constructor of the convolutional layer. One such hyperparameter is the number of filters and the other is the weight initialization scheme. Because our algorithm discovers and trains a number of filters, we need neither initialize nor train the filters of the convolutional layer of the Keras model via backpropagation. We set the number of filters parameter as the number of filters η^L discovered by our algorithm. The weights of the convolutional layer are then set with the output of our algorithm $\mathbf{W^L}$. We configure the trainable parameter of the convolutional layer $\mathbf{L}$ as False to prevent further training of the convolutional layer in the Sequential model. If the model requires a max-pooling layer following the convolutional layer, a MaxPool2D layer is added to the Sequential model.

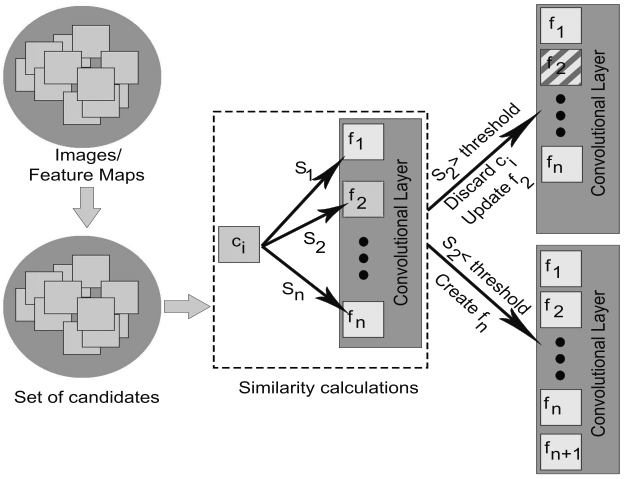

FIGURE 1. The proposed filter extraction process. When filter candidate $\mathbf{C_i}$ is selected from the set of candidates, similarity scores between $\mathbf{C_i}$ and all previously discovered filters of the current convolutional layer are calculated. In the figure, the highest similarity is between candidate $\mathbf{C_i}$ and filter f_2. If the similarity is greater than the similarity threshold, filter f_2 is updated and the candidate $\mathbf{C_i}$ is discarded. Otherwise, candidate $\mathbf{C_i}$ becomes a new filter for this convolutional layer.

The process of filter discovery is slightly different for the subsequent convolutional layers. In the deeper layers, instead of the training images, feature maps are processed to form the candidate set $\mathbf{C}$. The remainder of the filter discovery process is the same.

IV. IMPLEMENTATION AND EXPERIMENTS

The proposed method is applied to different CNN models to evaluate its feature extraction performance. The experiments are conducted with handwritten character and digit datasets MNIST [49], EMNIST [50], Kuzushiji-MNIST [51], and fashion items dataset Fashion-MNIST [52].

A. EXPERIMENTAL SETUP

Experiments are conducted using models with different architectures, as shown in Table 1. All models in our experiments are implemented using Keras with Theano [53] backend. Aside from the configuration of convolutional layers, Keras handles the execution of convolution and max-pooling

TABLE 1. CNN model types that are used in experiments. Convolutional layers are shown with filter size $n \times n$ where n is one of 3 or 5. Max pooling layers apply maxpooling operation to feature maps on $m \times m$ windows with strides of 2 where m is 2.

Model Type	Feature Extractor Layer 1	Layer 2	Layer 3	Layer 4	Layer 5	Classifier
A	Conv $n \times n$	MaxPool $m \times m$	Conv $n \times n$	-	-	Dense (1000), Dropout (50%), Dense (500), Output (10)
B	Conv $n \times n$	Conv $n \times n$	MaxPool $m \times m$	-	-	
C	Conv $n \times n$	MaxPool $m \times m$	Conv $n \times n$	MaxPool $m \times m$	Conv $n \times n$	
D	Conv $n \times n$	Conv $n \times n$	MaxPool $m \times m$	Conv $n \times n$	Conv $n \times n$	

operations on GPU. On all models, the fully connected layers are trained by Keras. ReLU is chosen as the activation function for all layers except the last one. Categorical cross-entropy is chosen for the evaluation of the loss, and Adadelta [54] is used for weight updates on fully connected layers. All convolutional layer filter weights are set with the output of our proposed method, and they are frozen (i.e., untrainable) so that they cannot be trained anymore by Keras. The fully connected layers are trained for 50 epochs. Experiments are run on a desktop computer using a GTX1050 GPU with 2GBs of VRAM and a 3.6 GHz Intel Core i7 7700 CPU. Further implementation details of the models used in the experiments is shared in the Appendix A.

B. DATASETS

The following datasets are used in the experiments to evaluate the performance of the proposed approach without applying any preprocessing or data augmentation on them.

1) MNIST

MNIST is a handwritten digit dataset. It consists of 60000 training and 10000 test images. For our experiments, the training set is divided into 50000 training and 10000 validation images. The validation set is randomly selected and separated from the original training set. The distribution of the classes is unbalanced in both the training and validation sets.

2) EMNIST-DIGITS

Extended Modified NIST (EMNIST) is an extended version of the MNIST dataset, which includes both handwritten digits and letters gathered from the NIST Special Database 19 dataset. It is organized into different subsets such as EMNIST-Letters, EMNIST-Digits, and EMNIST-Balanced. We use EMNIST-Digits subset in our experiments. The EMNIST-Digits subset contains 240000 training and 40000 test samples. The last 40000 images of the training set has already been arranged as a validation set [50]. The distribution of the classes is balanced in the preseparated validation set.

3) KUZUSHIJI-MNIST

Kuzushiji MNIST is a dataset for old cursive Japanese that contains ten different hiragana classes. In cursive Japanese, each hiragana has more than one drawing style because they are derived from different kanjis. Thus, each class is represented by more than one character with a completely different drawing style. It is a challenging dataset compared to the original MNIST dataset with extensive intraclass variations. The training, validation, and test set image counts are the same as the original MNIST. The validation set is randomly selected and separated from the original training set. Thus, the distribution of classes is unbalanced in both training and validation sets.

4) FASHION-MNIST

Fashion-MNIST is an MNIST-like dataset which contains images of fashion products. The dataset provides a training set of 50000 images and a test set of 10000 images. We partition the dataset into training, validation, and test sets applying the same procedure used on the MNIST dataset.

C. PERFORMANCE METRICS

To assess the performance of the proposed method, we calculated accuracy (4), precision (5), recall (6), specificity (7) and F1-score (8) for each class of the given dataset per model. To calculate these metrics for $class_i$, we assume that

- ***True Positive (TP):*** number of images that belong to $class_i$ and are correctly identified as $class_i$;
- ***True Negative (TN):*** number of images that belong to other classes and are correctly identified as other classes;
- ***False Positive (FP):*** number of images that belong to other classes but are incorrectly identified as $class_i$;
- ***False Negative (FN):*** number of images that belong to $class_i$ but incorrectly identified as other classes.

We also report the overall accuracy of all models.

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \times 100 \tag{4}$$

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{5}$$

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{6}$$

$$\text{specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}} \tag{7}$$

$$\text{F1-score} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{8}$$

D. RESULTS

Experiments are conducted by tuning the similarity threshold to evaluate the performance of our proposed method. The value of the similarity threshold is selected from the interval $[0, 1]$, which corresponds to 0%-100% similarity. The threshold values are selected with steps of 0.1 using grid search to find the best threshold values for convolutional layers. The values below 0.5 for the similarity threshold do not perform well because the number of filters extracted from

TABLE 2. Number of 5 × 5 filters extracted in each layer and classification accuracy of CNN models on the MNIST dataset with different similarity thresholds. Only the best performance is reported for each model.

Model Type	Similarity Threshold (T)				Filter Count (FC)				Accuracy(%)
	T1	T2	T3	T4	FC1	FC2	FC3	FC4	
A	0.6	0.5	-	-	97	54	-	-	**99.19**
B	0.6	0.5	-	-	97	117	-	-	99.18
C	0.6	0.6	0.7	-	97	120	67	-	98.34
D	0.6	0.5	0.6	0.7	97	117	101	145	97.45

TABLE 3. Confusion matrix of Model type A for MNIST dataset.

		Predicted Labels									
		0	1	2	3	4	5	6	7	8	9
Actual Labels	0	974	1	1	0	0	0	3	1	0	0
	1	0	1130	1	1	0	0	3	0	0	0
	2	1	1	1025	0	1	0	0	3	1	0
	3	0	0	0	1002	0	2	0	0	4	2
	4	0	0	1	0	974	0	2	0	0	5
	5	2	0	0	4	0	883	2	1	0	0
	6	3	3	0	0	1	2	949	0	0	0
	7	0	2	2	0	0	0	0	1022	1	1
	8	1	0	3	2	0	2	0	1	963	2
	9	0	2	1	2	3	1	0	2	1	997

the datasets decreases significantly. We avoid preprocessing or data augmentation of any kind to assess our approach in isolation. Each model is trained five times and the best results are presented in the following sections.

1) PERFORMANCE ON MNIST

We apply various combinations of similarity thresholds to each model, as shown in Table 1. In Table 2, we show the similarity threshold value used for the first convolutional layer as T1, the similarity threshold value used for the first convolutional layer as T2, etc. The number of filters extracted for the first convolutional layer when the similarity threshold value T1 is used is shown in the table with FC1. We use the same analogy for the remaining convolutional layer filter counts as FC2, FC3, etc.

We observe that the 5 × 5 filters provide higher accuracy values in the experiments compared to the smaller 3×3 filters. Model type A achieved the best classification performance on the MNIST dataset with 99.19% accuracy, as shown in Table 2. The filter extraction process for the convolutional layers and training of the best-performing model are completed within 30 minutes.

Our model makes 81 incorrect predictions out of 10000 test images. The confusion matrix for the model is presented in Table 3. We observe that the best predicted digit class is 1, whereas the most confusing class is digit 9, as presented in Table 4.

2) PERFORMANCE ON EMNIST-DIGITS

We observe that using a filter size of 5 × 5 results in higher accuracy values in the EMNIST-Digit experiments as in the MNIST experiments. The best performing model is again Model type A with 99.39% accuracy on the test set, as shown in Table 5.

TABLE 4. Performance metrics of Model type A for MNIST dataset.

Digits	Accuracy(%)	Precision	Recall	Specificity	F1-score
0	99.87	0.9929	0.9939	0.9992	0.9934
1	99.86	0.9921	0.9956	0.9990	0.9938
2	99.84	0.9913	0.9932	0.9990	0.9923
3	99.83	0.9911	0.9921	0.9990	0.9916
4	99.87	0.9949	0.9919	0.9995	0.9934
5	99.84	0.9921	0.9899	0.9992	0.9910
6	99.81	0.9896	0.9906	0.9989	0.9901
7	99.86	0.9922	0.9942	0.9991	0.9932
8	99.82	0.9928	0.9887	0.9992	0.9907
9	99.78	0.9901	0.9881	0.9989	0.9891

TABLE 5. Number of 5 × 5 filters that are extracted in each layer and classification accuracy of CNN models on the EMNIST-Digits dataset with different similarity thresholds. Only the best performance is reported for each model.

Model Type	Similarity Threshold (T)				Filter Count (FC)				Accuracy(%)
	T1	T2	T3	T4	FC1	FC2	FC3	FC4	
A	0.6	0.5	-	-	145	116	-	-	**99.39**
B	0.5	0.5	-	-	78	161	-	-	99.38
C	0.6	0.5	0.7	-	145	116	101	-	99.11
D	0.5	0.5	0.6	0.5	78	161	148	114	98.94

TABLE 6. Confusion matrix of Model type A for EMNIST-Digits dataset.

		Predicted Labels									
		0	1	2	3	4	5	6	7	8	9
Actual Labels	0	3984	2	3	0	0	1	7	0	1	2
	1	1	3983	8	0	1	0	3	3	1	0
	2	3	3	3982	4	1	0	0	2	4	1
	3	1	0	8	3962	0	14	0	4	6	5
	4	2	1	1	0	3972	0	4	3	0	17
	5	5	1	1	4	0	3970	9	0	5	5
	6	6	1	0	0	5	3	3985	0	0	0
	7	0	1	7	1	3	0	0	3981	0	7
	8	0	7	8	3	2	3	5	1	3960	11
	9	0	1	0	2	4	3	0	7	3	3980

Our model incorrectly predicts 244 of 40000 test images. A confusion matrix is presented for this model in Table 6. The best predicted class is digit 6, whereas digit 8 is the most confusing class for the model, as shown in Table 7.

3) PERFORMANCE ON KUZUSHIJI-MNIST

We observe that selecting a filter size of 3 × 3 yields better results in the Kuzushiji-MNIST dataset compared to the filter size of 5 × 5 used in the MNIST and EMNIST-Digit datasets. The best performance is observed on model type B, with an accuracy of 95.03%(see Table 8).

The total number of errors in the test set prediction is 497. A confusion matrix for the best model is presented in Table 9. The best predicted class is class 3, whereas class 2 is the most confusing character for our model, with a recall of 0.91, as can be seen in Table 10.

TABLE 7. Performance metrics of Model type A for EMNIST-Digits dataset.

Digits	Accuracy(%)	Precision	Recall	Specificity	F1-score
0	99.92	0.9955	0.9960	0.9995	0.9958
1	99.92	0.9958	0.9958	0.9995	0.9958
2	99.87	0.9910	0.9955	0.9990	0.9933
3	99.87	0.9965	0.9905	0.9996	0.9935
4	99.89	0.9960	0.9930	0.9996	0.9945
5	99.87	0.9940	0.9925	0.9993	0.9933
6	99.89	0.9930	0.9963	0.9992	0.9946
7	99.90	0.9950	0.9953	0.9994	0.9951
8	99.85	0.9950	0.9900	0.9994	0.9925
9	99.83	0.9881	0.9950	0.9987	0.9915

TABLE 8. Number of 3 × 3 filters extracted in each layer and classification accuracy of CNN models on Kuzushiji-MNIST dataset for different similarity thresholds. The best performance is reported for each model.

Model Type	Similarity Threshold (T)				Filter Count (FC)				Accuracy(%)
	T1	T2	T3	T4	FC1	FC2	FC3	FC4	
A	0.6	0.5	-	-	51	48	-	-	94.62
B	0.6	0.5	-	-	51	67	-	-	**95.03**
C	0.6	0.6	0.5	-	51	133	43	-	94.90
D	0.6	0.5	0.5	0.4	51	67	140	193	93.55

TABLE 9. Confusion matrix of Model type B for Kuzushiji-MNIST dataset.

Actual Labels \ Predicted Labels	0	1	2	3	4	5	6	7	8	9
0	952	3	2	1	23	4	0	9	5	1
1	0	946	8	0	7	3	17	2	8	9
2	8	7	911	44	6	6	5	2	5	6
3	2	1	10	979	1	0	3	1	2	1
4	11	11	1	10	939	3	7	4	9	5
5	1	6	23	5	3	943	6	1	5	7
6	4	3	12	8	4	1	964	2	1	1
7	3	1	6	1	8	1	7	959	2	12
8	1	10	1	10	7	2	2	0	966	1
9	6	5	8	1	9	0	6	2	19	944

TABLE 10. Performance metrics of Model type B for Kuzushiji-MNIST dataset.

Classes	Accuracy(%)	Precision	Recall	Specificity	F1-score
0	99.16	0.9636	0.9520	0.9960	0.9578
1	98.99	0.9527	0.9460	0.9948	0.9493
2	98.40	0.9277	0.9110	0.9921	0.9193
3	98.99	0.9245	0.9790	0.9911	0.9510
4	98.71	0.9325	0.9390	0.9924	0.9357
5	99.23	0.9792	0.9430	0.9978	0.9608
6	99.11	0.9479	0.9640	0.9941	0.9559
7	99.36	0.9766	0.9590	0.9974	0.9677
8	99.10	0.9452	0.9660	0.9938	0.9555
9	99.01	0.9564	0.9440	0.9952	0.9502

4) PERFORMANCE ON FASHION-MNIST

We observe that the experiments achieved better results with a filter size of 3×3 that with 5×5 filters. The best-performing model is Model type B with 90.11% accuracy, as shown in Table 11.

TABLE 11. Number of 3 × 3 filters extracted in each layer and classification accuracy of CNN models on the Fashion-MNIST dataset for different similarity thresholds.

Model Type	Similarity Threshold (T)				Filter Count (FC)				Accuracy(%)
	T1	T2	T3	T4	FC1	FC2	FC3	FC4	
A	0.7	0.7	-	-	92	48	-	-	88.80
B	0.7	0.6	-	-	92	40	-	-	**90.11**
C	0.7	0.7	0.6	-	92	48	44	-	85.55
D	0.7	0.7	0.5	0.5	92	62	47	18	86.92

TABLE 12. Confusion matrix of Model B for Fashion-MNIST dataset. The classes are numbered from 0 to 9. The class labels in order are Tshirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle boot.

Actual Labels \ Predicted Labels	0	1	2	3	4	5	6	7	8	9
0	877	1	6	21	1	1	87	0	5	1
1	3	976	0	14	2	0	3	0	2	0
2	23	0	807	10	85	0	72	0	3	0
3	23	7	9	918	22	0	17	0	4	0
4	0	1	88	42	805	0	62	0	2	0
5	0	0	0	1	0	977	0	17	0	5
6	121	1	55	21	56	0	735	0	11	0
7	0	0	0	0	0	8	0	972	0	20
8	1	0	3	7	1	0	2	4	982	0
9	0	0	0	0	0	5	1	32	0	962

The lowest performance is obtained on the Fashion-MNIST dataset with the Shirt class as shown in Table 12. We observe that articles of clothing that belong to the Shirt class are often confused with the T-shirt/Top, Coat, Pullover, and Dress classes because of interclass similarity in the dataset samples. The best predicted class is Bag, with a 98.2% correct prediction rate, as shown in Table 13.

E. DISCOVERED FILTERS

1) MNIST

The filters extracted from the MNIST dataset in the first layer of Model B are shown in Fig. 2. Directed edges and curves can be observed in the extracted filters. The filters obtained using our algorithm are meaningful representations of the visual cues in the dataset. An interesting observation is that some of our features converged to Gabor-like filters, which have been frequently used in the literature.

2) EMNIST-DIGITS

The EMNIST dataset is an extended version of the MNIST dataset. This is why some of the filters extracted from the EMNIST-Digits training set (see Fig. 3) are the same or very similar to those shown in Fig. 2.

3) KUZUSHIJI-MNIST

In the Kuzushiji-MNIST experiments, we observe higher classification accuracy by extracting 3×3 filters compared to extracting 5×5 filters. The filters extracted from the training set for the first convolutional layer are illustrated in Fig. 4.

TABLE 13. Performance metrics of Model type B for Fashion-MNIST dataset.

Classes	Accuracy(%)	Precision	Recall	Specificity	F1-score
Tshirt/top	97.06	0.8368	0.8770	0.9810	0.8565
Trouser	99.66	0.9899	0.9760	0.9989	0.9829
Pullover	96.46	0.8337	0.8070	0.9821	0.8201
Dress	98.02	0.8878	0.9180	0.9871	0.9027
Coat	96.38	0.8282	0.8050	0.9814	0.8164
Sandal	99.63	0.9859	0.9770	0.9984	0.9814
Shirt	94.91	0.7508	0.7350	0.9729	0.7428
Sneaker	99.19	0.9483	0.9720	0.9941	0.9600
Bag	99.55	0.9732	0.9820	0.9970	0.9776
Ankle boot	99.36	0.9737	0.9620	0.9971	0.9678

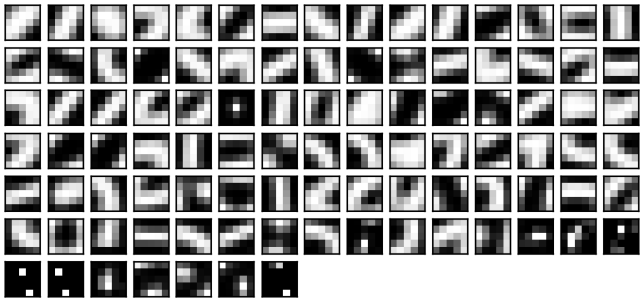

FIGURE 2. The filters discovered from the MNIST training images using our unsupervised algorithm. All filters are 5 × 5 and the similarity threshold is 0.6 for the first convolutional layer as shown in Table 2.

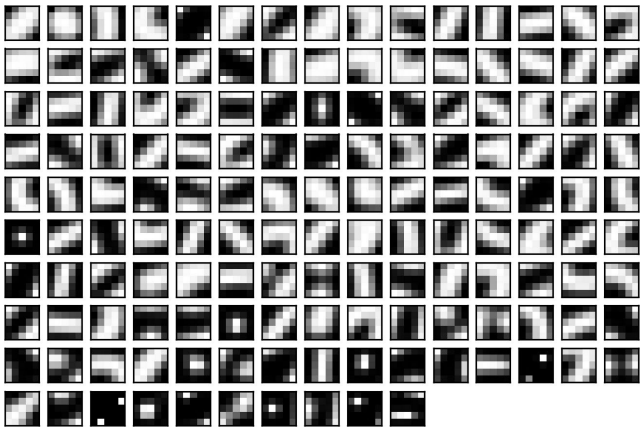

FIGURE 3. The filters discovered from the EMNIST-Digits training images using our unsupervised algorithm. All filters are 5 × 5 and the similarity threshold is 0.6 for the first convolutional layer as shown in Table 4.

Directed edges and parts of curved strokes can be observed from the extracted filters.

4) FASHION-MNIST

The filters extracted from the Fashion-MNIST dataset contains features such as directed edges and corners. We have more filters that capture smooth curves in the MNIST and EMNIST-Digits datasets because many digits have more curve like features compared to articles of clothing in the Fashion-MNIST dataset. The extracted filters are shown in Fig. 5.

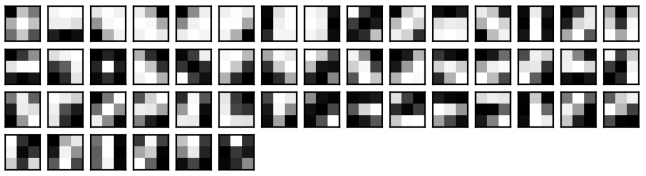

FIGURE 4. The filters discovered from the Kuzushiji-MNIST training images using our unsupervised algorithm. All filters are 3 × 3 and the similarity threshold is 0.6 for the first convolutional layer as shown in Table 6.

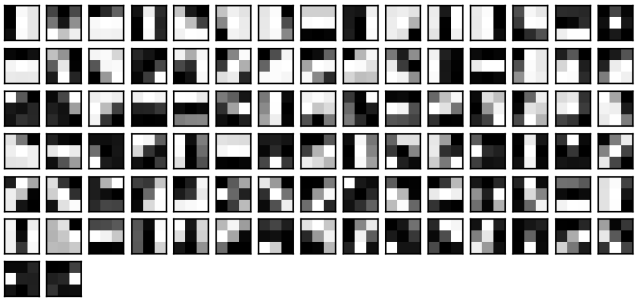

FIGURE 5. The filters discovered from the Fashion-MNIST training images using our unsupervised algorithm. All filters are 3 × 3 and the similarity threshold is 0.7 for the first convolutional layer as shown in Table 8.

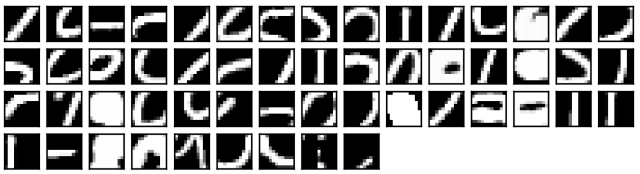

FIGURE 6. Visualization of the 54 features extracted in the second layer from the MNIST training images using Model A.

5) FILTERS IN DEEPER LAYERS

The first convolutional layer filters are easy to plot, as their input weights represent specific features and can be simply reshaped to reconstruct images. However, beyond the first layer, the weights are not directly mapped to the input pixels. Thus, a method is implemented to visualize the features in deep layers to obtain a better understanding of the features extracted by our algorithm.

After the training is complete, we feed the training set to the trained model again to obtain the feature maps for each image at the specified layer. The pixel with the highest value among all the feature maps is marked for each training image. The coordinates of this pixel are traced back to the original training image. The region is marked in the original image contained the feature that stimulates this specific filter the most. This is depicted in Fig. 6 using the MNIST dataset. Fig. 6 suggests that the filters are specialized to seek features that gradually evolve to represent more complex features. These complex features represent parts of the digits such as closed loops and curves that are present in digits such as 9 or 4.

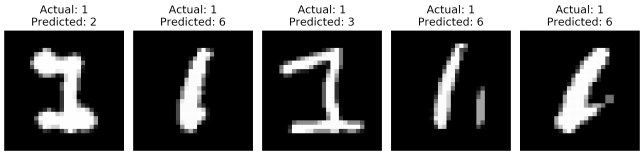

FIGURE 7. Test images from MNIST dataset's digit class 1 that are incorrectly labeled by Model A. The second, fourth, and fifth images are classified as 6 due to the artifacts and the curves present in the images.

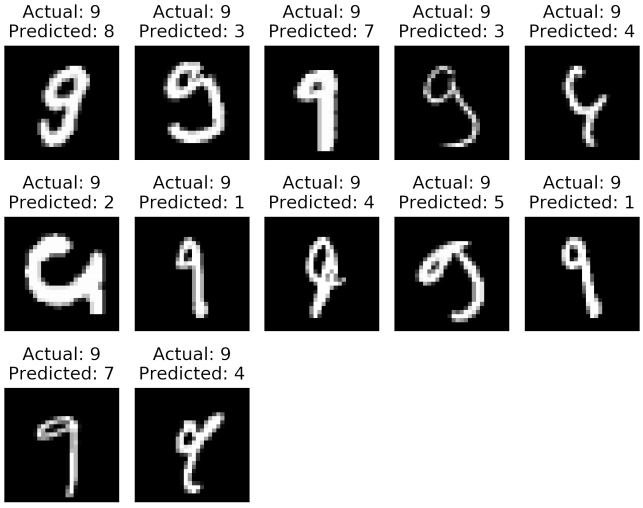

FIGURE 8. Test images from MNIST dataset's digit class 9 that are incorrectly labeled by Model A.

F. MISCLASSIFIED SAMPLES

1) MNIST

For the MNIST dataset, our model performs nearly perfectly in labeling digit 1 samples in the test set by making only 5 incorrect predictions out of 1135 digit 1 samples.

The mispredicted digit 1 samples can be seen in Fig. 7. The second, fourth, and fifth images are classified as digit 6. We observe that the slight angle and curve in the digit strokes and the artifacts in the samples can be the reasons that mislead the prediction. The first and third misclassified images are easy to label for human observer, whereas the trained model predicts them as digits 2 and 3, respectively. We observe some base features that are also present in the other digit 2 samples in these two test samples, which can easily sway the decision towards an incorrect classification. When the top-2 predictions for each of these test samples are checked, we observe that the second most likely prediction is digit 1 with a very close confidence level to the top-1 prediction.

Our model's lowest prediction score belongs to digit 9 for the MNIST test set. Our model mislabels 12 digit 9 images (see Fig. 8). Our model tends to vote for different digits based on different styles. When the loop diameter is small, the image is either labeled as digit 1 or 7 based on the length of the loop because the loop feature becomes indiscernible or completely lost during the convolution and pooling operations. The first image in the second row shows an interesting

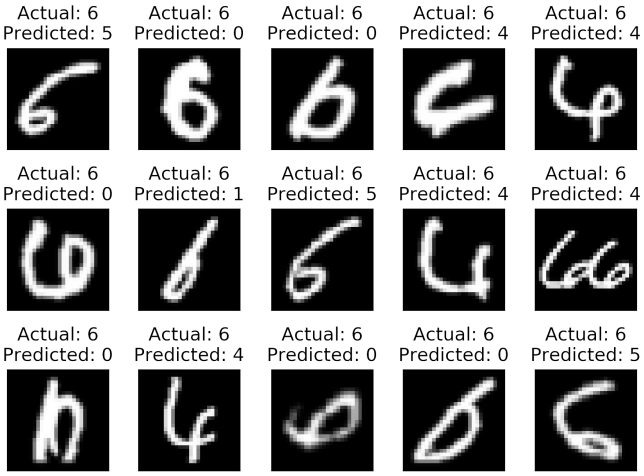

FIGURE 9. Test images from EMNIST-Digits dataset's digit class 6 that are incorrectly labeled by Model A.

case. The bottom half of the digit is cropped, rendering the image unrecognizable. All digit 4 predictions are made for unusual digit 9 samples. Only one of them is recognized as digit 9 by a human observer. Even though our model's top-1 prediction for these samples are incorrect, the next most likely prediction for 8 of these samples are digit 9.

2) EMNIST-DIGITS

The best prediction accuracy is obtained from digit 6 samples of the EMNIST-Digits test set. The erroneously predicted samples are shown in Fig. 9. Among the 4000 digits 6 test samples, 15 incorrect predictions are made by our model. We observe that some of the mispredicted samples do not resemble digit 6 at all. One of the samples actually has a two-digit number 66 instead of digit 6 in it. We observe that rotation and missing parts due to cropping influence our model to favor digit 4 during prediction. Aside from digit 4, digit 0 is a common misprediction. When top-2 predictions are observed, digit 6 is the next prediction in 12 of 15 cases.

The worst prediction performance is observed for the digit 8 class. Our model mislabels 40 of the 4000 digit 8 images from the EMNIST-Digits test set. Some of the mispredicted images in Fig. 10 are missing digit parts that would help correct the prediction. We observe that digit 8 is more frequently predicted as digit 9. When inspected, it can be seen that 4 of those images do not have a loop in the lower half of digit 8. The common feature in the wrong predictions is the lack of proper curves that the model is expecting for a digit 8 sample to have. Some of the samples do not even resemble digit 8. The second best prediction for 22 of the incorrectly predicted images are digit 8.

3) KUZUSHIJI-MNIST

In the Kuzushiji-MNIST experiments, our best model achieves the best classification performance on class 3 with 21 errors in prediction. We observe that class 3 samples are often confused with class 2 samples and are labeled as

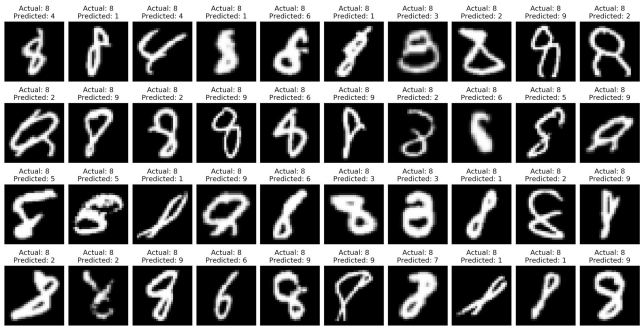

FIGURE 10. Test images from EMNIST-Digits dataset's digit class 8 that are incorrectly labeled by Model A.

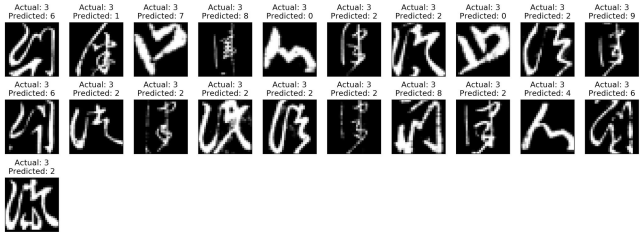

FIGURE 11. Test images from Kuzushiji-MNIST dataset's class 3 that are incorrectly labeled by Model B.

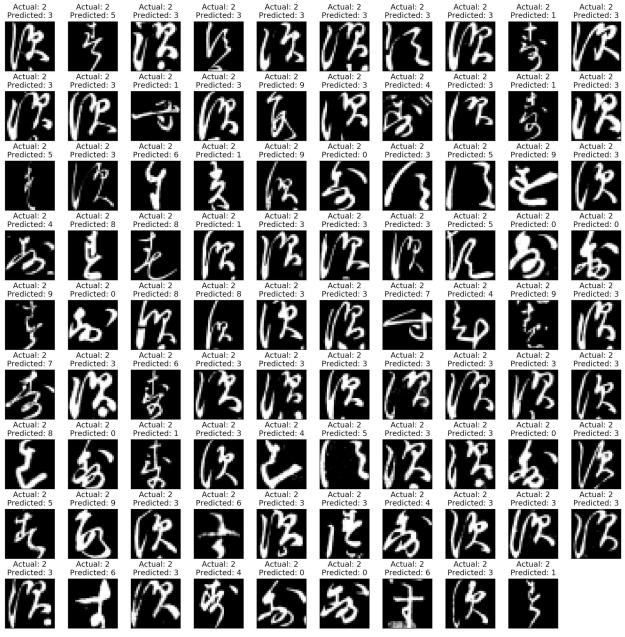

FIGURE 12. Test images from Kuzushiji-MNIST dataset's class 2 that are incorrectly labeled by Model B.

such at the test time. Some of the images in Fig. 11 have features similar to those of the other classes, which make the prediction more difficult. The second best prediction for the 13 of the incorrectly predicted images are class 3.

The most confusing class is 2 for our model. Class 2 images are often labeled as class 3 by our model. When we examine the test images that are mislabeled as class 3, we observe that

FIGURE 13. Test images that belonged to Bag class but incorrectly labeled as different classes by our trained model B.

FIGURE 14. Test images that belonged to Shirt class but incorrectly labeled with similar classes by model B.

most of them contain features similar to those of the class 2 samples (Fig. 12). The second best prediction for 55 of these samples is correct.

4) FASHION-MNIST

Our model performs best when Bag class samples are encountered. 18 out of 1000 bag test images are incorrectly predicted. In Fig. 13, incorrectly labeled test samples belonging to Bag class can be observed. For example, the second image in the first row is predicted as a Pullover sample. The two long sleeve-like parts of the bag image could be the misleading features in the prediction in this example. Bag class images are most commonly confused with the Dress class images in our model. When the second-top predictions are checked for these images, only 2 are correctly predicted. Because the Fashion-MNIST images are created by down sampling colored fashion article images into the MNIST format, most of the details/features of objects are lost. We think that some of the errors in the test set can be avoided if the images are at a higher resolution.

The Shirt class is the most confusing class for our model with 265 misclassified test samples. Some of the incorrectly predicted Shirt images from the test class are shown in Fig. 14. The Shirt samples in the test set are commonly mistaken for T-shirt/Top class samples. When closely inspected, it can be observed that our model learned to recognize the fashion article in the image as T-shirt/Top when short or no sleeves are present. When the top-2 predictions are inspected, 196 of the 265 are correctly predicted as Shirt.

V. DISCUSSION

Our method trains the convolutional layers in an unsupervised scheme without using backpropagation, while training the fully connected layers in a supervised manner. Pre-

TABLE 14. Comparison of the number of epochs of training needed for convolutional filters, whether data augmentation and ensemble of networks are used. The legend to read the table: ✓: applied, ×: not applied, NA: no information available.

Method	Data Augmentation	Ensemble	Backpropagation	Epochs
HVC [55]	✓	✓	✓	300
DropConnect [56]	✓	✓	✓	1000
MCDNN [57]	✓	✓	✓	800
OptConv+Log+Perc [58]	✓	×	✓	1000
CAMNet3 [59]	✓	✓	✓	NA
SAM [60]	✓	×	✓	NA
CAE [34]	×	×	✓	NA
Deep k-Sparse AE + F.T. [37]	NA	×	✓	200
SPC-best ensemble [29]	×	✓	✓	NA
SPC-best single [29]	×	×	✓	NA
k-Sparse AE [37]	NA	×	✓	5000
Disentangled [61]	×	×	✓	50
Ours	×	×	×	1
Ours ensemble	×	✓	×	1
Ours init. + train	×	×	×(init.) + ✓(train)	1 (init.) + 50 (train)

vious studies either trained the network end-to-end in a supervised manner, unsupervised manner with pseudo-label backpropagation, or a mix of unsupervised feature learning for initialization using supervised backpropagation. Compared with supervised methods, we do not need any labels to train the convolutional layers because no backpropagation is used in the training. In addition, our method can extract filters without prior domain knowledge, in contrast to self-supervised learning methods, where handcrafting of pretext tasks requires domain knowledge to perform well. Our method may appear similar to unsupervised pre-training, which is used only to initialize the network weights. However, we do not use extracted filters for initialization, because we avoid supervised training in the convolutional layers.

A. COMPARISON TO OTHER WORKS

We compare the performance of the unsupervised (Table 15), mixed (Table 16), and supervised (Table 17) methods with those of our proposed method. Other methods employ data augmentation, ensembles, and a large number of training epochs along with backpropagation training to achieve better results, as summarized in Table 14.

1) COMPARISON TO UNSUPERVISED WORKS

The state-of-the-art classification accuracy of unsupervised methods for the MNIST dataset is 99.21% [29], as shown in Table 15. This accuracy is obtained from a clustering setting that utilizes an ensemble of 15 AEs to form clusters. These clusters are related to k-sets of pseudo-labels. A consensus function selects only the points that receive the same pseudo-label in all k-sets for the pseudo-label training of an MLP. The strength of this method comes from the ensemble of AEs. When only a single AE is used instead of an ensemble, the accuracy decreases to 98.02%, which is worse than that of our proposed method. For a fair comparison, we build an ensemble that includes top-3 performing Model type A networks that we obtain from running the proposed method. This ensemble performs better than [29], with 99.28% accuracy on the test set. The k-sparse AE [37] discusses an unsupervised training scheme in which the extracted features are fixed, and a logistic regression classifier is trained using these features. However, this unsupervised method achieves an accuracy score of only 98.65% on the MNIST dataset.

TABLE 15. Comparison of the proposed model with other unsupervised methods in the literature.

Method	MNIST	F-MNIST
SPC-best ensemble [29]	99.21	67.94
SPC-best single [29]	98.02	59.23
k-sparse AE [37]	98.65	-
Ours	99.19	**90.11**
Ours ensemble	**99.28**	**90.43**

TABLE 16. Comparison of the proposed model with other mixed methods in the literature.

Method	MNIST	F-MNIST
CAE [34]	99.29	-
Deep k-Sparse AE + F.T. [37]	99.03	-
Disentangled [61]	96.20	85.60
Ours	99.19	**90.11**
Ours init. + train	**99.43**	**91.93**

The best unsupervised classification accuracy on the Fashion-MNIST dataset belongs to SPC-best with 67.94% accuracy. Other unsupervised methods are not shown in Table 15 because the rest of the literature attains lower accuracy.

To our knowledge, no unsupervised study has been conducted with either EMNIST-Digits or Kuzushiji-MNIST datasets in the literature.

2) COMPARISON TO MIXED METHODS

Masci et al. [34] train a CAE for unsupervised feature extraction, and those features are used to initialize a CNN. This CNN is subsequently trained end-to-end in a supervised manner. The classification accuracy of this method is 99.29%. This method may resemble our proposed method; however, after extracting features in an unsupervised manner, we neither use the features for initialization nor train them further. A similar method [37] achieves an accuracy of 99.03%. The features are extracted using a sparsity constraint on the AE and fine-tuned in a supervised manner. For a fair comparison between our proposed method and mixed methods [34], [37], we use the output of our algorithm for convolutional filter initialization and fine-tuned the CNN model type A (see Table 1) with backpropagation. We report this model's performance as "Ours init. + train" in Table 16. We surpass the performance of the mixed methods with 99.43% accuracy on the MNIST test set.

The best mixed-method performance for Fashion-MNIST is reported in [61]. They achieve 85.60% accuracy by training AE and applying k-means clustering based on soft nearest neighbor loss, which requires data labels. The performance of "Ours init. + train" for the Fashion-MNIST dataset exceeds the performance of [61] by 6.33% without data augmentation.

TABLE 17. Comparison of the proposed model with other supervised methods in the literature.

Method	MNIST	EMNIST-Digits	K-MNIST	F-MNIST
HVC [55]	99.83	-	-	93.89
DropConnect [56]	99.79	-	-	-
DropConnect no aug. [56]	99.43	-	-	-
MCDNN 35-net [57]	99.77	-	-	-
MCDNN 1-net [57]	99.53	-	-	-
OptConv+Log+Perc [58]	-	99.43	-	-
CAMNet3 [59]	99.78	-	99.05	94.34
CAMNet3 no aug. [59]	99.47	-	97.48	93.00
SAM [60]	-	-	-	96.41
Ours	99.19	99.39	95.03	90.11
Ours init. + train	99.43	**99.63**	96.48	91.93

3) COMPARISON TO SUPERVISED WORKS

The state-of-the-art for MNIST is 99.83%, which is obtained by training a capsule networks in a supervised manner [55]. We share this result only to demonstrate the best classification accuracy achieved for MNIST among all the methods. Our architecture does not involve capsules and is not similar to [55] to compare. Architecturally similar methods DropConnect [56] and MCDNN [57], both report results obtained from an ensemble of networks with the help of data augmentation. When no data augmentation is applied, the accuracy of DropConnect's [56] 5-network ensemble drops to 99.43% with 1000 epochs of training. As opposed to this result, our model offers a good alternative with a simple architecture (i.e. no ensemble) which is much faster to train (only a single epoch to train convolutional layers) and achieves 99.19% accuracy. When we apply our method to the MNIST dataset and use the extracted filter weights to initialize a single Model Type A network, we observe that our network performs the same compared to the 5-network ensemble of DropConnect without using data augmentation on the MNIST dataset with only 50 epochs of training.

The state-of-the-art performance on the EMNIST-Digits dataset is 99.43% with the supervised OptConv+Log+Perc [58] method, in which a large optical convolution with logarithmic activation is applied before perceptron training on the images. The experiment in [58] requires a special camera setup to offer the best result, whereas we work on raw dataset images. In contrast to our study, [58] applies data augmentation to the training images. Our best-performing model achieves 99.39% accuracy on the EMNIST-Digits test set, as listed in Table 5. This performance score is close to the current best performance result. For a fair comparison, when we further train the extracted filters, we observe that the accuracy of our model outperforms that of [58] with 99.63% accuracy.

The best performance on the Kuzushiji-MNIST dataset is observed on model B, with an accuracy of 95.03% (see Table 8), which is close to the simple CNN performance of 95.12% in the original Kuzushiji-MNIST paper [51]. Compared to the MNIST dataset, there are significant intraclass variations in Kuzushiji-MNIST. Most of the samples belonging to the same class do not resemble each other. This renders the classification of this dataset a difficult task without data augmentation. The state-of-the-art accuracy for this dataset is 99.05% using CAMNet3 [59]. This architecture is quite different from our architecture. CAMNet3 is a multipath CNN architecture in which data flow is routed to one of the parallel networks based on the content of the images. Data flow routing between the parallel networks in CAMNet3 allows the intraclass variation in Kuzushiji-MNIST to be learned better than the classic CNN architecture used in our experiments. When we use the extracted filters for initialization of the convolutional layers and apply backpropagation, the performance of the model increases to 96.48% without data augmentation.

Compared to MNIST, Fashion-MNIST has more complex features, more intraclass variations, and interclass similarities. Thus, we expect a decrease in the classification performance compared to that of MNIST. The state-of-the-art accuracy for the Fashion-MNIST dataset is obtained from a supervised network, with an accuracy of 96.41% [60]. Wide-Res-Net-28-10 [62] and Shake-Shake [63] regularization along with data augmentation methods are used in [60].

VI. CONCLUSION

We propose an unsupervised and backpropagationless training algorithm for training the convolutional layers of CNNs. Our algorithm performs training by extracting new features from a training set without using label information. The entire filter extraction process is unsupervised. Conventionally, the correct number of filters for the convolutional layer must be determined using hyperparameter optimization techniques. However, the hyperparameter that defines the number of neurons is no longer needed to be determined by applying hyperparameter optimization tecniques because it is automatically determined by our algorithm at the end of the filter extraction process. The self-discovery of filters does not require a weight initialization mechanism. Thus, we can skip the selection of a suitable initialization algorithm for the convolutional layer weights based on the given dataset, model, or activation function.

We obtain promising results on different datasets without the aid of data preprocessing, augmentation, or carefully constructed complex architectures. We demonstrate that it is possible to train convolutional layers through a single pass on training set images using an unsupervised backpropagationless approach, as opposed to thousands of iterations required in other studies in the literature. Although our performance on different data sets appears to be lower than that of supervised methods, the results we obtain results that are comparable to the state-of-the-art using a much simpler and easily trainable model. When our model is adjusted to compete more fairly with supervised studies, we even see it performs on par or better compared to some supervised works using the MNIST or EMNIST-Digits datasets. When compared to other unsupervised or mixed works on even ground, our method performs with higher accuracy.

TABLE 18. Architectural details of CNN Model type A built for MNIST dataset.

Layers	Input	Output
conv_1[5, 97]	(batch_size, 1, 28, 28)	(batch_size, 97, 28, 28)
max_pooling	(batch_size, 97, 28, 28)	(batch_size, 97, 14, 14)
conv_2[5, 54]	(batch_size, 97, 14, 14)	(batch_size, 54, 14, 14)
flatten	(batch_size, 54, 14, 14)	(batch_size, 10584)
dense_1	(batch_size, 10584)	(batch_size, 1000)
dropout[50%]	(batch_size, 1000)	(batch_size, 1000)
dense_2	(batch_size, 1000)	(batch_size, 500)
dense_3	(batch_size, 500)	(batch_size, 10)

TABLE 19. Architectural details of CNN Model type A built for EMNIST-Digits dataset.

Layers	Input	Output
conv_1[5, 145]	(batch_size, 1, 28, 28)	(batch_size, 145, 28, 28)
max_pooling	(batch_size, 145, 28, 28)	(batch_size, 145, 14, 14)
conv_2[5, 116]	(batch_size, 145, 14, 14)	(batch_size, 116, 14, 14)
flatten	(batch_size, 116, 14, 14)	(batch_size, 22736)
dense_1	((batch_size, 22736)	(batch_size, 1000)
dropout[50%]	(batch_size, 1000)	(batch_size, 1000)
dense_2	(batch_size, 1000)	(batch_size, 500)
dense_3	(batch_size, 500)	(batch_size, 10)

TABLE 20. Architectural details of CNN Model type B built for Kuzushiji-MNIST dataset.

Layers	Input	Output
conv_1[3, 51]	(batch_size, 1, 28, 28)	(batch_size, 51, 28, 28)
conv_2[3, 67]	(batch_size, 51, 28, 28)	(batch_size, 67, 28, 28)
max_pooling	(batch_size, 67, 28, 28)	(batch_size, 67, 14, 14)
flatten	(batch_size, 67, 14, 14)	(batch_size, 13132)
dense_1	(batch_size, 13132)	(batch_size, 1000)
dropout[50%]	(batch_size, 1000)	(batch_size, 1000)
dense_2	(batch_size, 1000)	(batch_size, 500)
dense_3	(batch_size, 500)	(batch_size, 10)

TABLE 21. Architectural details of CNN Model type B built for Fashion-MNIST dataset.

Layers	Input	Output
conv_1[3, 92]	(batch_size, 1, 28, 28)	(batch_size, 92, 28, 28)
conv_2[3, 40]	(batch_size, 92, 28, 28)	(batch_size, 40, 28, 28)
max_pooling	(batch_size, 40, 28, 28)	(batch_size, 40, 14, 14)
flatten	(batch_size, 40, 14, 14)	(batch_size, 7840)
dense_1	(batch_size, 7840)	(batch_size, 1000)
dropout[50%]	(batch_size, 1000)	(batch_size, 1000)
dense_2	(batch_size, 1000)	(batch_size, 500)
dense_3	(batch_size, 500)	(batch_size, 10)

Possible future directions for this work include improving the classification performance when there are too many variations within the same class, and extending this work to color images.

APPENDIX A
ARCHITECTURAL DETAILS OF THE MODELS

The number of filters and the weights of the filters obtained as the output of the proposed algorithm are used in the convolutional layers in all of the models used in our experiments. In all models, the convolutional layers are set as untrainable so that backpropagation training is not applied to them. The convolution operation is applied with a stride of 1 in each convolutional layer. The padding is set to be the same in the convolutional layers such that the output has the same size as the input for these layers. We set the use_bias parameter to false in the convolutional layers. Max pooling is applied to 2×2 windows with strides of 2. We use ReLU as the activation function in all convolutional and dense(fully connected) layers, except the last dense layer, where we use softmax. In all models, we have only one dropout layer that is configured to drop 50% (rate $= 0.5$) of the incoming connections randomly during the training of the dense layers. Only the dense layers are trained for 50 epochs with the Adadelta optimizer configured with default values (lr=1.0, rho=0.95, epsilon=None, decay=0) and the loss function categorical cross entropy.

In Tables 18-21, the convolutional layer receptive field size and the number of filters are shown as conv_[receptive field size, number of filters]. Dropout rate is shown as a percentage in square brackets. Input and output shapes are shown in channels-first fashion (batch_size, channels, rows, columns) for the convolutional and max pooling layers because we use Theano as the backend.

The ensembles that we use for comparisons are built using the top three best-performing models obtained by applying the proposed algorithm to the specified datasets. We compute the average of the outputs of the three networks to report the accuracy of the ensemble.

A. MNIST MODEL

See Table 18.

B. EMNIST-DIGITS MODEL

See Table 19.

C. KUZUSHIJI-MNIST MODEL

See Table 20.

D. FASHION-MNIST MODEL

See Table 21.

REFERENCES

[1] S. Linnainmaa, "Taylor expansion of the accumulated rounding error," *BIT Numer. Math.*, vol. 16, no. 2, pp. 146–160, Jun. 1976.
[2] J. Hochreiter, "Untersuchungen zu dynamischen neuronalen Netzen," Diploma thesis, Institut für Informatik, Tech. Univ. Munich, Munich, Germany, 1991.
[3] O. Sener and S. Savarese, "Active learning for convolutional neural networks: A core-set approach," in *Proc. 6th Int. Conf. Learn. Represent. (ICLR)* 2018, pp. 1–13. [Online]. Available: https://openreview.net/forum?id=H1aIuk-RW
[4] H. H. Aghdam, A. Gonzalez-Garcia, A. Lopez, and J. Weijer, "Active learning for deep detection neural networks," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*. Los Alamitos, CA, USA: IEEE Computer Society Press, Oct. 2019, pp. 3671–3679.
[5] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 44–436, May 2015.
[6] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," *J. Mach. Learn. Res.*, vol. 9, pp. 249–256, May 2010.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, L. O'Conner, Ed. Los Alamitos, CA, USA: IEEE Computer Society Press, Dec. 2015, pp. 1026–1034.

[8] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian optimization of machine learning algorithms," in *Proc. Adv. Neural Inf. Process. Syst. 26th Annu. Conf. Neural Inf. Proc. Syst.*, vol. 25, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Red Hook, NY, USA: Curran Associates, 2012, pp. 2951–2959.

[9] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, Feb. 2012.

[10] I. Ilievski, T. Akhtar, J. Feng, and C. A. Shoemaker, "Efficient hyperparameter optimization of deep learning algorithms using deterministic RBF surrogates," in *Proc. 31st AAAI Conf. Artif. Intell.* Los Alamitos, CA, USA: AAAI Press, 2017, pp. 822–829.

[11] E. Bochinski, T. Senst, and T. Sikora, "Hyper-parameter optimization for convolutional neural network committees based on evolutionary algorithms," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Sep. 2017, pp. 3924–3928.

[12] R. Andonie and A.-C. Florea, "Weighted random search for CNN hyperparameter optimization," *Int. J. Comput. Commun. Control*, vol. 15, no. 2, pp. 74–80, Mar. 2020.

[13] A. Dosovitskiy, J. T. Springenberg, M. Riedmiller, and T. Brox, "Discriminative unsupervised feature learning with convolutional neural networks," in *Proc. 27th Int. Conf. Neural Inf. Process. Syst.*, vol. 1, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Cambridge, MA, USA: MIT Press, 2014, pp. 766–774.

[14] C. Doersch, A. Gupta, and A. A. Efros, "Unsupervised visual representation learning by context prediction," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*. L. O'Conner, Ed. Los Alamitos, CA, USA: IEEE Computer Society Press, Dec. 2015, pp. 1422–1430.

[15] M. Noroozi and P. Favaro, "Unsupervised learning of visual representations by solving jigsaw puzzles," in *Computer Vision—ECCV*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham, Switzerland: Springer, 2016, pp. 69–84.

[16] R. Zhang, P. Isola, and A. A. Efros, "Colorful image colorization," in *Computer Vision—ECCV*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham, Switzerland: Springer, 2016, pp. 649–666.

[17] G. Larsson, M. Maire, and G. Shakhnarovich, "Learning representations for automatic colorization," in *Computer Vision—ECCV*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham, Switzerland: Springer, 2016, pp. 577–593.

[18] G. Larsson, M. Maire, and G. Shakhnarovich, "Colorization as a proxy task for visual understanding," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*. L. O'Conner, Ed. Los Alamitos, CA, USA: IEEE Computer Society Press, Jul. 2017, pp. 840–849.

[19] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros, "Context encoders: Feature learning by inpainting," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*. L. O'Conner, Ed. Los Alamitos, CA, USA: IEEE Computer Society Press, Jun. 2016, pp. 2536–2544.

[20] I. Misra, C. L. Zitnick, and M. Hebert, "Shuffle and learn: Unsupervised learning using temporal order verification," in *Computer Vision—ECCV*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham, Switzerland: Springer, 2016, pp. 527–544.

[21] X. Wang, K. He, and A. Gupta, "Transitive invariance for self-supervised visual representation learning," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*. L. O'Conner, Ed. Los Alamitos, CA, USA: IEEE Computer Society Press, Oct. 2017, pp. 1338–1347.

[22] J. Cho, Y. Kim, H. Jung, C. Oh, J. Youn, and K. Sohn, "Multi-task self-supervised visual representation learning for monocular road segmentation," in *Proc. IEEE Int. Conf. Multimedia Expo (ICME)*, L. O'Conner, Ed. Los Alamitos, CA, USA: IEEE Computer Society Press, 2018, pp. 1–6.

[23] X. Liu, J. V. D. Weijer, and A. D. Bagdanov, "Exploiting unlabeled data in CNNs by self-supervised learning to rank," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 8, pp. 1862–1878, Aug. 2019.

[24] J. Macqueen, "Some methods for classification and analysis of multivariate observations," in *Proc. 5th Berkeley Symp. Math. Statist. Probab.*, 1967, pp. 281–297.

[25] J. Yang, D. Parikh, and D. Batra, "Joint unsupervised learning of deep representations and image clusters," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society Press, Jun. 2016, pp. 5147–5156.

[26] R. Liao, A. Schwing, R. Zemel, and R. Urtasun, "Learning deep parsimonious representations," in *Proc. 30th Int. Conf. Neural Inf. Process. Syst.*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds. Red Hook, NY, USA: Curran Associates, 2016, pp. 5083–5091.

[27] J. Xie, R. Girshick, and A. Farhadi, "Unsupervised deep embedding for clustering analysis," in *Proc. 33rd Int. Conf. Mach. Learn.*, vol. 48, M. F. Balcan and K. Q. Weinberger, Eds. New York, NY, USA: PMLR, 2016, pp. 478–487.

[28] Y. Xu and R. McCord, "CoSTA: Unsupervised convolutional neural network learning for spatial transcriptomics analysis," *BMC Bioinform.*, vol. 22, no. 1, p. 397, Aug. 2021.

[29] L. Mahon and T. Lukasiewicz, "Selective pseudo-label clustering," in *KI 2021: Advances in Artificial Intelligence*, S. Edelkamp, R. Möller, and E. Rueckert, Eds. Cham, Switzerland: Springer, 2021, pp. 158–178.

[30] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*. L. O'Conner, Ed. Los Alamitos, CA, USA: IEEE Computer Society Press, Jun. 2016, pp. 770–778.

[31] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.

[32] X. Yan, I. Misra, A. Gupta, D. Ghadiyaram, and D. Mahajan, "ClusterFit: Improving generalization of visual representations," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society Press, Jun. 2020, pp. 6508–6517.

[33] S. Gidaris, P. Singh, and N. Komodakis, "Unsupervised representation learning by predicting image rotations," 2018, *arXiv:1803.07728*.

[34] J. Masci, U. Meier, D. Cireşan, and J. Schmidhuber, "Stacked convolutional auto-encoders for hierarchical feature extraction," in *Artificial Neural Networks and Machine Learning—ICANN*, T. Honkela, W. Duch, M. Girolami, and S. Kaski, Eds., vol. 6791. Berlin, Germany: Springer, 2011, pp. 52–59.

[35] B. Hou and R. Yan, "Convolutional auto-encoder based deep feature learning for finger-vein verification," in *Proc. IEEE Int. Symp. Med. Meas. Appl. (MeMeA)*, Jun. 2018, pp. 1–5.

[36] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *J. Mach. Learn. Res.*, vol. 11, pp. 3371–3408, Mar. 2010.

[37] A. Makhzani and B. J. Frey, "*k*-sparse autoencoders," in *Proc. 2nd Int. Conf. Learn. Represent.*, Y. Bengio and Y. LeCun, Eds., Banff, AB, Canada, Apr. 2014, pp. 1–9.

[38] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Proc. 27th Int. Conf. Neural Inf. Process. Syst.*, vol. 2, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, Eds. Cambridge, MA, USA: MIT Press, 2014, pp. 2672–2680.

[39] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, "InfoGAN: Interpretable representation learning by information maximizing generative adversarial nets," in *Proc. 30th Int. Conf. Neural Inf. Process. Syst.*, D. D. Lee, U. Luxburg, R. Garnett, M. Sugiyama, and I. Guyon, Eds. Red Hook, NY, USA: Curran Associates, 2016, pp. 2180–2188.

[40] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," in *Proc. 4th Int. Conf. Learn. Represent. (ICLR)*, Y. Bengio and Y. LeCun, Eds., San Juan, PR, USA, May 2016.

[41] Z. Ren and Y. J. Lee, "Cross-domain self-supervised multi-task feature learning using synthetic imagery," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.* Los Alamitos, CA, USA: IEEE Computer Society Press, Jun. 2018, pp. 762–771.

[42] K. Fukushima, "Neocognitron for handwritten digit recognition," *Neurocomputing*, vol. 51, pp. 161–180, Apr. 2003.

[43] K. Fukushima, "Training multi-layered neural network neocognitron," *Neural Netw.*, vol. 40, pp. 18–31, Apr. 2013.

[44] X. Wang and A. Gupta, "Unsupervised learning of visual representations using videos," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, L. O'Conner, Ed. Los Alamitos, CA, USA: IEEE Computer Society Press, Dec. 2015, pp. 2794–2802.

[45] D. H. Hubel and T. N. Wiesel, "Receptive fields of single neurones in the cat's striate cortex," *J. Physiol.*, vol. 148, no. 3, pp. 574–591, Oct. 1959.

[46] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biol. Cybern.*, vol. 36, no. 4, pp. 193–202, Apr. 1980.
[47] T. Erkoç and M. Taner Eskl, "Unsupervised similarity based convolutions for handwritten digit classification," in *Proc. 30th Signal Process. Commun. Appl. Conf. (SIU)*, May 2022, pp. 1–4.
[48] F. Chollet. (2015). *Keras*. [Online]. Available: https://github.com/fchollet/keras
[49] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
[50] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, "EMNIST: Extending MNIST to handwritten letters," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*. Anchorage, AK, USA: IEEE, May 2017, pp. 2921–2926.
[51] T. Clanuwat, M. Bober-Irizar, A. Kitamoto, A. Lamb, K. Yamamoto, and D. Ha, "Deep learning for classical Japanese literature," 2018, *arXiv:1812.01718*.
[52] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms," 2017, *arXiv:1708.07747*.
[53] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions," 2016, *arXiv:1605.02688*.
[54] M. D. Zeiler, "ADADELTA: An adaptive learning rate method," 2012, *arXiv:1212.5701*.
[55] A. Byerly, T. Kalganova, and I. Dear, "No routing needed between capsules," *Neurocomputing*, vol. 463, pp. 74–80, Nov. 2021.
[56] L. Wan, M. Zeiler, S. Zhang, Y. LeCun, and R. Fergus, "Regularization of neural networks using dropconnect," in *Proc. 30th Int. Conf. Mach. Learn.*, vol. 28, S. Dasgupta and D. McAllester, Eds. Atlanta, GA, USA: JMLR, 2013, pp. 1058–1066.
[57] D. Ciresan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.* Los Alamitos, CA, USA: IEEE Comput. Soc. Press, Jun. 2012, pp. 3642–3649.
[58] P. Pad, S. Narduzzi, C. Kündig, E. Türetken, S. A. Bigdeli, and L. A. Dunbar, "Efficient neural vision systems based on convolutional image acquisition," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, L. O'Conner, Ed. Los Alamitos, CA, USA: IEEE Computer Society Press, Jun. 2020, pp. 12282–12291.
[59] D. Tissera, K. Kahatapitiya, R. Wijesinghe, S. Fernando, and R. Rodrigo, "Context-aware multipath networks," 2019, *arXiv:1907.11519*.
[60] P. Foret, A. Kleiner, H. Mobahi, and B. Neyshabur, "Sharpness-aware minimization for efficiently improving generalization," in *Proc. 10th Int. Conf. Learn. Represent. (ICLR)*, 2021, pp. 1–9. [Online]. Available: https://openreview.net/forum?id=6Tm1mposlrM
[61] A. Fred Agarap and A. P. Azcarraga, "Improving k-means clustering performance with disentangled internal representations," 2020, *arXiv:2006.04535*.
[62] S. Zagoruyko and N. Komodakis, "Wide residual networks," 2016, *arXiv:1605.07146*.
[63] X. Gastaldi, "Shake-shake regularization," 2017, *arXiv:1705.07485*.

TUĞBA ERKOÇ received the B.S. and M.S. degrees in computer engineering from Işık University, Istanbul, Turkey, in 2006 and 2010, respectively, where she is currently pursuing the Ph.D. degree in computer engineering.

From 2010 to 2015, she was a Software Engineer with Nortel Networks Netaş, Istanbul. Since 2015, she has been a Research Assistant with the Computer Engineering Department, Işık University. Her research interests include image classification and object detection using machine learning techniques.

MUSTAFA TANER ESKİL received the B.S. degree in mechanical engineering and the M.S. degree in systems and control engineering from Boğaziçi University, Istanbul, Turkey, in 1997 and 1999, respectively, and the Ph.D. degree in computer science and engineering from Michigan State University, East Lansing, MI, in 2005.

He is currently a Professor and a Supervisor of the Pattern Recognition and Machine Intelligence Laboratory, Işık University. His research interests include image processing and machine learning, specifically facial expression analysis and synthesis, and object detection and segmentation.