Received 27 March 2023, accepted 11 April 2023, date of publication 17 May 2023, date of current version 25 May 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3277392

RESEARCH ARTICLE

Deep Learning Model for Recognition of Handwritten Devanagari Numerals With Low Computational Complexity and Space Requirements

DANVEER RAJPAL AND AKHIL RANJAN GARG, (Member, IEEE)
Department of Electrical Engineering, MBM University, Jodhpur 342011, India
Corresponding author: Danveer Rajpal (danveer.rajpal@rediffmail.com)

ABSTRACT The broad application area and accompanying challenges make machine learning-based recognition of handwritten scripts a demanding field. Individuals' writing practices and inherent variations in the size, shape, and tilt of characters may increase the difficulty level. Deep convolutional neural network (DCNN) models have been successful in solving pattern recognition problems, but at the expense of a considerable number of trainable parameters and heavy computational loads. The proposed work addresses these problems by using the shifted window (SWIN) transformer method to recognize handwritten Devanagari numerals for the first time. In the presented model, the SWIN transformer is finely tuned to withstand popular DCNN models, such as VGG-16Net, ResNet-50, and DenseNet-121, in terms of recognition accuracy, space requirement, and computational complexity. The model successfully attained a recognition accuracy of 99.20% with only 0.218 million trainable parameters and 0.0912 giga floating-point operations per second (FLOPs). This indicates the validity and soundness of the proposed model for recognizing handwritten Devanagari numerals.

INDEX TERMS Computational complexity, DCNN, devanagari numerals, DenseNet-121, ResNet-50, shifted window transformer, space complexity, VGG-16Net.

I. INTRODUCTION

In India, the Devanagari script has the ancient legacy of being the most extensively used script. Scholars have been researching machine learning-based recognition of handwritten Devanagari characters for more than 45 years [1]. About 528 million Indians consider Hindi (Devanagari) their primary language. They prefer it for reading and writing, as per the most recent language census [2] conducted in 2011 and released in 2018. Additionally, the number of Hindi users is increasing at a rate of 25% per decade. These statistics have motivated researchers to develop further automated language-processing tools related to the Devanagari script. The inherent variation in writing styles, character scaling, character skew, distorted forms of the characters, and uneven patterns due to pen, page, and paper bases are known issues related to machine-based recognition. Furthermore, similar strokes and the richness of the curves presented in the Devanagari script leverage the recognition complexity. The broad application area includes machine-based processing of bank checks, identification of pin codes from postal envelopes, conversion of handwritten documents into machine-readable forms, and the development of assistive technology for the disabled. Figure 1 illustrates a set of Devanagari numerals.

The three primary stages of machine-based recognition models related to the proposed problem are pre-processing character images, extracting salient features, and classifying patterns using the appropriate machine-learning algorithm. The feature extraction stage involves capturing salient

The associate editor coordinating the review of this manuscript and approving it for publication was Mounim A. El Yacoubi.

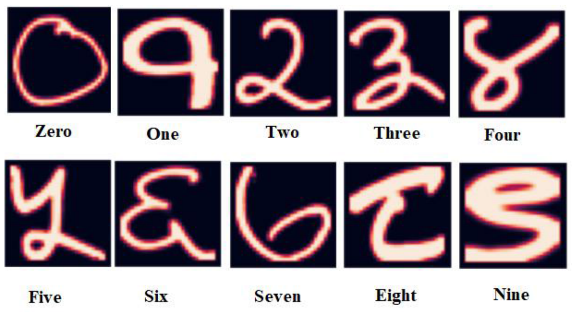


FIGURE 1. Set of handwritten devanagari-numerals.

features from character images. Structural and statistical features have been used to recognize Devanagari characters. The former are vertical-horizontal-slanted lines, curves, loops, intersections, endpoints, dots, strokes, contours, and directional details, whereas the latter are distance-based, zoning-based, moment-based, pixel-density-based, transformation-based, and histogram-based features. Various classification algorithms used in related studies can be broadly classified as ANN [3], kernel-based [4], statistical-based [5], template-matching-based [6], and fuzzy-logic-based [7].

Recent studies have used deep convolutional neural networks for recognition, which do not require much pre-processing or manual feature extraction. By achieving excellent recognition accuracy, these models become game changers, leaving no space for any significant leap in this regard. The main concerns related to deep convolutional neural network (DCNN) models are the need for large datasets, millions of trainable parameters, and demanding computational complexities.

Implementing deep neural networks in embedded systems is challenging because they require large memory and enormous computations [8]. Complex networks have placed significant demand on energy-constrained hardware platforms intended for model execution [9]. Several factors make it crucial to develop and deploy models with low computational complexity and space requirements [10], [11]. This can result in:

Efficient Deployment: Deep learning models are frequently implemented in embedded systems, field-programmable gate arrays (FPGA), application-specific integrated circuits (ASICs), mobile devices, and computers, which have limited power budgets and computational resources. Low-complexity models can be deployed more effectively on such devices.

Cost-Saving: Deep learning models require several computational resources and accelerators, such as hyperthreading, computer unified device architecture (CUDA) cores, higher bandwidth synchronous dynamic random-access memory (SDRAM), central processing units (CPUs), and graphical processing units (GPUs), which may be expensive. Devising lower-complexity models can result in cost savings for both researchers and businesses.

Improved Performance: Deep learning models are prone to overfitting because of the millions of trainable parameters, resulting in poor generalization to newer datasets. Limiting the number of parameters is a better solution. Furthermore, models with a lower computational complexity can result in faster responses.

These facts have inspired us to present a solution with lower complexity for the proposed problem.

The Vision-Transformer (VIT) model [12] has been introduced in natural language processing to address the concern of model complexity. Instead of considering the entire image at once, VIT uses the self-attention approach to assess the relationship across pixel pairs within short portions of an image. This results in a computational cost reduction of up to the extreme levels. The cost-effectiveness of VIT makes it popular in the computer vision domain for various applications such as the processing of high-definition images [13], pattern recognition [14], and pattern segmentation [15]. According to current developments in the computer vision field, convolutional neural networks (CNN) may be replaced by vision transformers in relevant application areas. VIT is appropriate for mobile platforms such as field-programmable gate arrays (FPGA) and Raspberry Pi owing to their minimal space and computational complexity requirements [16].

The main objective of the proposed work is to address the problem of machine-based recognition of the Devanagari script through the vision transformer concept and investigate the performance of the model against popular pre-trained DCNN models in terms of recognition accuracy, space complexity, and computational complexity.

A. RELATED WORK

Significant work has been recorded on the machine-learning-based recognition of the English script owing to its global presence, whereas the language-based automation related to the Hindi (Devanagari) script is still in its early stages. This section covers the benchmarking models based on the proposed problem. Bajaj et al. [17] created features based on the pixel density, moments (left, right, top, and bottom), and descriptive components (various strokes). They employed a hybrid classifier consisting of a multi-layer perceptron (MLP), Kohonen-Net, Kohonen-self organizing maps (SOM), and meta-pi networks for numeral classification. Bhattacharya et al. [18] retrieved horizontal and vertical stroke-based features from numeral patterns to construct shape vectors. They estimated Bayesian posterior probabilities using the MLP and hidden Markov model (HMM) networks separately, and combined the results for the final classification through the MLP network. Hanmandlu et al. [19] segmented individual numeral patterns into small boxes to collect features in the form of normalized vector distances. They prepared fuzzy sets from these features and applied a modified exponential membership function for the classification. Lakshmi et al. [20] evaluated the features of the segmented blocks of numerical images in terms of the gradient direction. The feature size was constrained using principal component analysis (PCA) method, followed by the development of histograms. They applied distance-based matching for classification. Several nearest-neighbor-based classification algorithms, including nearest-neighbor (NN), k-nearest-neighbor (KNN), Euclidian-distance-based KNN, cosine-similarity-based KNN, condensed-NN, reduced-NN, farthest-like-neighbor, and nearest-unlike-neighbor were the subject of comparative research by Holambe et al. [21]. They obtained curvature and gradient-based features from numeral images for classification. Das et al. [22] combined the features based on quad-tree-longest-run and modular PCA to gather local and global details regarding Devanagari numerals. They employed a one-versus-all support vector machine (SVM) classifier for the recognition task. Iamsa et al. [23] examined the effectiveness of an extreme learning machine (ELM) and a deep learning-based feedforward-backpropagation neural network (DFBNN) over features related to the histogram of gradient (HOG). Jarunthai et al. [24] observed the performance of generalized radial basis function-based modified ELM for numeral recognition. They selected the centers of radial basis function (RBF) kernels using a semi-optimized method. Khanduja et al. [25] created hybrid feature vectors by combining the structural (endpoints, intersection points, and loops) and statistical (pixel distribution) features. They applied hybrid vectors to a feedforward neural network to recognize Devanagari numerals. Chaurasia et al. [26] modified the architecture of a conventional CNN by replacing the classification module (dense layers) with an SVM to exploit its structural risk minimization potential. The authors recorded an improvement in the recognition of the Devanagari digits. Rajpal et al. [27] obtained features from numeral patterns using DCNN models (VGG-16Net and VGG-19Net). After reducing the feature size, they combined two types of features into a single vector. They employed an MLP network for the classification. Garg et al. [28] optimized the features received from DenseNet-121 for Devanagari digits. They implemented the PCA method to eliminate feature co-linearity and applied the resultant along with statistical metrics (skewness, kurtosis, and variance) to train and test the MLP classifier.

Other significant works on the handwritten Devanagari script are as follows. Acharya et al. [29] applied the layer-wise dropout function and data augmentation to limit the overfitting of the proposed DCNN model. They improved the performance of the model by increasing the size of the dataset. Chakraborty et al. [30] observed an improvement in the results of a proposed CNN model with increasing depth. They also examined a hybrid model consisting of a CNN and bidirectional long short-term memory (BLSTM) for the same classification problem; however, it could not surpass the base CNN model. Jangid et al. [31] examined the performance of the proposed DCNN model with six different optimizers: adaptive moment (ADAM), stochastic gradient descent (SGD), adaptive gradient (ADAGRAD), adaptive delta (ADADELTA), adaptive maximum (ADAMAX), and root mean square propagation (RMSPROP). The authors introduced the concept of layer-wise training. Sonawane et al. [32] employed a pretrained AlexNet model for the recognition task. They obtained comparable results, with fewer training samples and shorter processing times. Aneja et al. [33] presented a comparative study of pre-trained DCNN models: AlexNet, DenseNet-121, DenseNet-201, VGG-11, VGG-16, VGG-19, and Inception-v3 in the recognition of Devanagari scripts. They analyzed the results in terms of time complexity and recognition accuracy. Guha et al. [34] presented a CNN model with optimized time and space complexities. They examined it against popular DCNN models: LeNet 5, ResNet (18, 34, and 50), AlexNet, Inception-v3, and DenseNet-121. They observed that a smaller network might perform well with proper optimization of the hyperparameters. Bhati et al. [35] examined the performance of VGG-16Net and DenseNet-121 with deep fine-tuning and shallow tuning in the recognition of the Devanagari script. Individual models were trained by unfreezing a varying number of trainable layers. DenseNet-121, with deep fine-tuning, is the top performer. Rajpal et al. [36] created fusion-based hybrid feature vectors that included the features received from VGG-19Net, Resnet-50, Inception-v3, and discrete-wavelet-transform. They used hybrid features to train and test the SVM for classifying Devanagari scripts.

Some excellent work related to other important scripts is as follows: Alrobah et al. [37] developed a hybrid deep model consisting of CNN, SVM, and gradient-boosting classifiers for the recognition of Arabic script. Rasheed et al. [38] fine-tuned the hyperparameters of a pre-trained DCNN model, AlexNet, for the recognition of a handwritten Urdu script. Li et al. [39] developed a customized CNN model for recognizing handwritten Chinese scripts. This model can expand the dataset to improve the recognition rate. Aly et al. [40] introduced a deep convolutional self-organizing maps model for recognizing handwritten English digits. Gupta et al. [41] developed a multilingual digit recognition model that includes English, Hindi, Bangla, Telugu, Arabic, Odia, Gujarati, and Punjabi scripts. In their model, the features received from the CNN outperformed the handcrafted features.

Researchers have applied several strategies to different benchmarking models, which has left no stone unturned in achieving a high recognition rate. This has raised the need to introduce some unique concepts to address the concern of model correctness by considering model complexity.

The majority of related models are based on conventional machine learning methods that have a limited ability to process raw natural data. These methods often require domain expertise and manual feature engineering to achieve reasonable results. The performance of these methods strongly depends on the design of the feature extractor. In contrast, deep learning models can automatically learn high-level features from raw data, eliminating the need for manual feature engineering. It is particularly useful in applications related to image, text, audio, and video processing, where the input data are high dimensional and complex [42]. This inspired us to examine deep learning methods for solving the proposed problem.

B. MOTIVATION

The richness of the curves and the resembling strokes in the Devanagari script make it typical for a language-based automation system. The related models achieved good accuracy levels using deep convolutional neural networks. The main concerns with these models are the requirement of a large number of trainable parameters that demand more space; the higher depth of the networks invites computational complexity in many folds; and the need for larger datasets (one of the probable examples is the ImageNet dataset) for their fruitful training, which may not be available for applications related to natural language processing and medical imaging. The main driving force behind the proposed method is to address these concerns by developing a low-complexity (space and computational) model with a comparable success rate.

1) CONTRIBUTION

A novel concept of vision-transformer-based learning has been adopted for the first time to solve the problem of machine-based recognition of handwritten Devanagari numerals. Efforts have been made to achieve a comparable recognition rate by addressing concerns about the space and computational complexities related to deep learning-based models. The proposed model incorporates a vision transformer with a shifted window to extract salient features and classify corresponding numeral patterns. The same recognition problem was solved using popular pre-trained DCNN models: VGG-16Net, ResNet-50, and DenseNet-121. For validation purposes, the results of the proposed model were compared with those of the DCNN models. Various performance metrics were computed and visualized to provide further insight into the proposed model.

The remainder of this paper is organized as follows: Section II covers the theoretical foundation of the model, Section III describes the experimental approach, Section IV presents the result analysis and critical discussions, and Section V concludes the paper.

II. PRELIMINARY

This section provides an overview of the methods used in this study. The methods used were the SWIN transformer and DCNN models VGG-16Net, ResNet-50, and DenseNet-121.

A. VISION TRANSFORMER

Convolutional or recurrent neural networks with an encoder and decoder are the foundation of frequently used sequence translation models. The top-performing models additionally use an attention mechanism to link the encoder and the decoder. Vaswani et al. [12] proposed a revolutionary architecture known as a transformer to simplify network design. It is entirely dependent on attention processes and has the potential to replace recurrences and convolutions. The enormous success of transformers in natural language processing has inspired researchers to solve pattern recognition problems by using them. It has been observed in some studies [15], [43] that, despite its potential, the vision transformer model struggles with high-definition images because of the limited resolution of the feature maps and the increased computational complexity of the order of quadratic to the input pattern size. Additionally, fixed-scale tokens of vision transformers are highly inadequate owing to the variable-scale visual features associated with vision applications. Liu et al. [44] addressed these issues by including the shifted window concept in the vision transformer, which can handle high-definition images while maintaining a check on the computational complexity. Using this concept, the model complexity becomes linearly related to the input image size. These facts motivated us to examine the performance of SWIN transformers in recognizing handwritten Devanagari numerals. Figure 2 illustrates the architecture of the SWIN transformer.

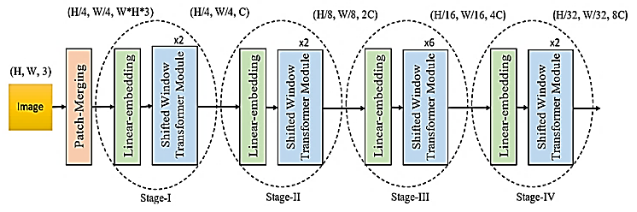


FIGURE 2. The architecture of shifted window transformer.

A SWIN transformer can act as a basic building block for computer vision applications. It is a hierarchical transformer that relies on a shifted window mechanism. In this scheme, self-attention estimation is bound to nonoverlapping windows with cross-window connections. This model is capable of handling information at different scales linear computational complexity concerning the size of the input image.

The Patch-Merging module and SWIN-transformer module are the two main components of the SWIN transformer architecture. The input image was divided into patches of size (n, n), and the patch-merging module arranged these patches depth-wise. This process results in the down-sampling of the input image by a factor of n. Consequently, the image size is rescaled from (H, W, C) to (H/n, W/n, n^2*C), where H, W, and C represent the height, width, and channels associated with the image, respectively. Figure 3 illustrates the complete patch-merging process.

The normalization layer, multi-head self-attention section with standard and shifted windows, and multilayer perceptron are the core components of the SWIN transformer module. Figure 4 depicts the arrangement of the components.

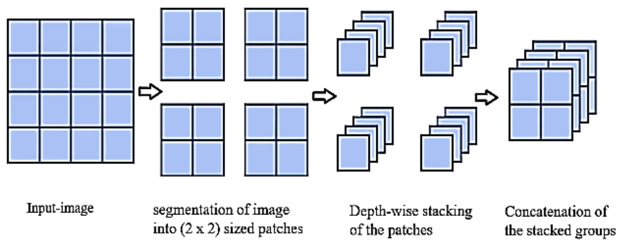

Input-image
segmentation of image into (2 x 2) sized patches
Depth-wise stacking of the patches
Concatenation of the stacked groups

FIGURE 3. The sequence of operations related to patch-merging.

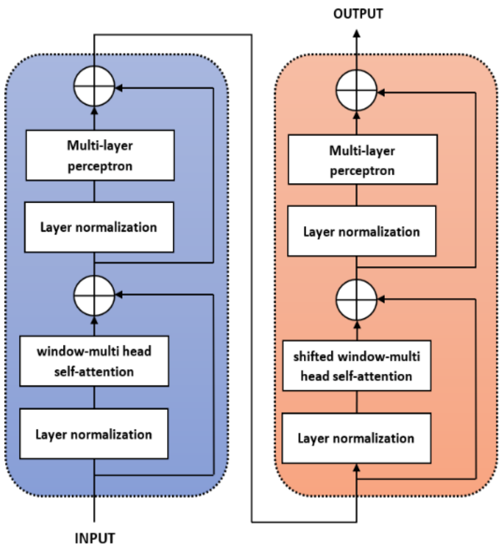


FIGURE 4. Two successive modules of swin transformer.

In the SWIN transformer module, a shifted window-based multi head self-attention block is followed by a 2-layer MLP with Gaussian error linear unit (GELU) nonlinearity. Layer normalization was applied before the individual units of the multi-head self-attention module and the MLP network. A residual connection is added to each module. In a window-based multi head self-attention scheme, the window size is kept constant throughout the network, which results in a linear complexity concerning the patch count.

Self-attention is limited to individual windows in a window-based scheme, which can severely limit the network's modelling capabilities. To address this concern, developers introduced a shifted-window self-attention mechanism. Figure 5 presents an overview of the shifted-window implementation.

The figure above shows that the window is moved diagonally towards the bottom right corner of an image by a value of M/2 for window size M. Shifting of the window results in non-overlapping patches that do not belong to any window and windows with incomplete patches, as depicted in Figure 5(b). The cyclic shift technique relocates non-overlapping patches into windows with incomplete patches, as shown in Figures 5 (c) and (d).

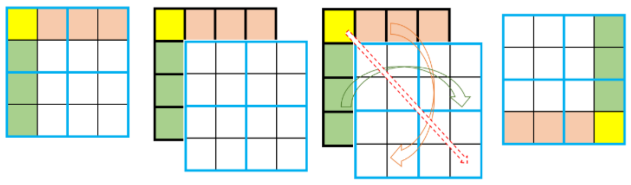


FIGURE 5. Shifting of window in shifting window multi-head self-attention scheme. (a) window of size (M, M); (b) window shifting by (M/2); (c) cyclic-shift of the non-overlapped patches; (d) completion of one shift.

During the calculation, a mask was applied to refocus the self-attention on the adjacent patches because, following this shift, a window may include patches that were not adjacent in the original feature map. This method establishes significant cross-connections between windows and has been proven to enhance system performance. Better computational complexity was recorded for shifted window self-attention than for conventional self-attention. The following expressions list the corresponding complexities.

$$\Omega\,(\mathrm{MSA}) = 4\mathrm{hwC}^2 + 2(\mathrm{hw})^2\mathrm{C} \tag{1}$$

$$\Omega\,(\mathrm{W-MSA}) = 4\mathrm{hwC}^2 + 2\mathrm{M}^2\mathrm{hwC} \tag{2}$$

where h, w, C, and M represent the height, width, channel count, and window size associated with the image, respectively. Based on Equations (1) and (2), it is clear that the complexity of window-based multihead self-attention (W-MSA) is linear for a fixed value of M, against the quadratic complexity of the conventional multihead self-attention (MSA).

B. VGG-16Net

Simonyan and Zisserman [45] of the Visual Geometric Group (VGG) at Oxford University created a network. The model comprised 16 trainable layers, of which 13 were convolutional and three were fully connected. The network participated in the ImageNet large-scale visual recognition challenge (ILSVRC) in 2014. The network was trained using the ImageNet dataset, which contains approximately 14 million photos connected to 1000 different item classes. The top-5 test-accuracy score of the model in the competition was 92.7%. VGG-16Net consists of five convolutional blocks, each carrying convolutional layers and a max-pooling layer, with the final blocks serving as classification blocks.

C. ResNet-50

ResNet, also known as the Residual Network [46], was introduced in the 2015 Image-Net competition. In the past, it was assumed that stacking additional convolutional layers would help extract more complicated features from given patterns, which would then increase the accuracy and robustness of the network. ResNet developers have noted that adding too many layers to the network begins to saturate and occasionally degrades its accuracy. Network initialization, exploding or vanishing gradients, and parameter optimization are the possible causes of this degradation. ResNet effectively addresses the issues of deep convolutional networks by introducing the crucial concept of residual learning. The ResNet-50 network has 50 trainable layers, 49 of which are convolutional layers, and one of which is a dense layer.

D. DenseNet-121

A densely connected network known as DenseNet [47] pioneered the idea of feature map reuse through dense connections. Deeper networks stack many layers between the network's input and output, which may cause the loss of information during the forward transit and the loss of gradient during backpropagation. The possible consequences include improper training and increased error rates. Another issue is the large number of trainable parameters associated with deeper networks, which may result in a heavy computational load. DenseNet can effectively handle these problems through its dense connections, where each layer is connected directly to every other layer in the network. DensNet is a faster, deeper, and more computationally efficient network. The network contains 121 trainable layers.

III. MATERIAL AND METHOD

The dataset of handwritten Devanagari numerals was compiled from a well-known repository made available to the researcher by Acharya et al. [29]. Developers have collected handwritten documents from different individuals belonging to various age groups, professions, and places. The authors scanned and cropped the images of individual characters manually and placed them in their respective folders. For the suggested study, 20,000 images of handwritten Devanagari numerals were collected from the indicated source.

The Python environment was used to construct the experimental framework. Several open-source libraries, including OpenCV, Python's imaging library, TensorFlow, Keras, NumPy, Pandas, Scikit-Learn, Seaborn, and Python's time module were used for the proposed setup. The simulations were performed on the robust Google Co-laboratory platform. The Tesla K-80 2496 CUDA cores, 12 GB of GDDR5-VRAM GPU, a hyper-threaded Xeon processor, 12.6 GB of RAM, and 33 GB of storage are all supported by the Co-laboratory.

The architecture of the SWIN transformer implemented in this study is shown in Figure 6.

The images of the dataset were resized as (32, 32, 1); that is, the width (W) and height (H) were both maintained at 32. In the first step, the 'Patch-segmentation' section of the vision transformer divides the input image into isolated, non-overlapping patches, as displayed in Figure 7.

The individual patch was considered a token and the associated feature was configured as a concatenation of the grey-level values related to each pixel. Because we utilized a patch size of (2, 2), the individual patch had a feature dimension of ($2 \times 2 \times 1 = 4$) in our implementation. The grey-level-valued feature was projected onto a certain dimension (represented as C) with the help of a linear-embedding module. The value of C was observed to be in the range of

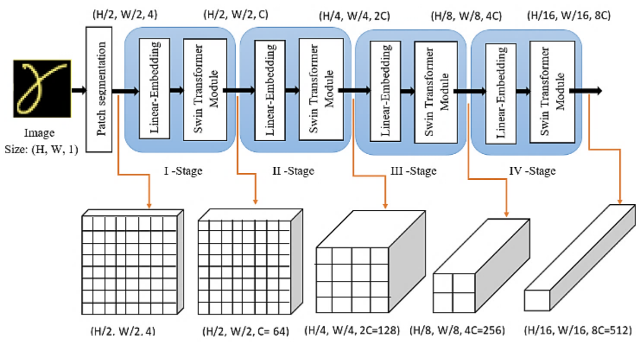


FIGURE 6. Layout of shifted window transformer implemented in the proposed work.

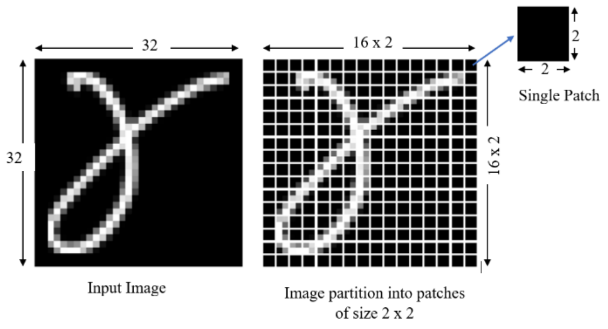


FIGURE 7. Patch segmentation of input image.

32 to 128 with a step size of 16 in the proposed implementation. The optimum result was obtained with C equal to 64. The patch tokens are processed according to the number of transformer modules with shifted window mechanisms. The token counts (H/2 $\times$ W/2) are conserved by the transformer modules. The transformer module with linear embedding is termed as stage I.

With increasing depth of the network, the token counts were reduced via patch-merging layers to maintain the hierarchical representation of the feature maps. The features associated with individual groups of (2×2) adjacent patches were concatenated by the first patch-merging layer. The features that were composed into 4C- dimensions were processed with a linear layer as well.

This resulted in a reduction of token counts by ($2 \times 2 = 4$) folds, and the dimension of the output was set to 2C. Later, shifted window transformer modules were employed for the transformation of the features by maintaining the feature map resolution as (H/4, W/4). Stage II refers to the initial phase of the patch merging and feature transformation. The process was then carried out two more times as stage III and stage IV, with the resultant output sizes being (H/8, W/8) and (H/16, W/16).

The dataset was split into training, testing, and validation sets in a ratio of 70:20:10. The validation set was used to adjust the reasonable values of various hyperparameters, which are listed in Table 1. The ranges of various hyperparameters that were adopted in the validation stage were as

TABLE 1. List of hyper-parameters used in the proposed experiment.

Parameter	Value used in the experiment
Patch size	(2, 2)
Dropout rate	0.03
No. of heads	8
Embedded dimension (C)	64
Learning rate	0.001
Batch size	128
No. of epochs	30
Weight decay	0.0001
Label smoothing	0.1
Optimizer	Adaptive moment (Adam)

follows: dropout rate (0.01 to 0.05), embedded-dimension (32 to 128 in steps of 16), learning rate (0.0008 to 0.005), weight decay (0.0001 to 0.0005), and label smoothing (0.05 to 0.3).

Various regularization techniques, such as weight dropout, weight decay, and label smoothing, were used in the proposed experiment to check for overfitting and enhance the accuracy of the model. The results were recorded in terms of the recognition accuracy, precision, recall, and F1-measure; these metrics are listed in Table 2.

The number of trainable parameters and floating-point operations per second (FLOPs) as estimated during the training phase to obtain a sense of the space requirements and computational complexity of the proposed model.

To validate the proposed model, the same dataset was applied to popular pre-trained deep learning models for classification, namely, VGG-16Net, ResNet-50, and DenseNet-121. Unlike the proposed SWIN transformer model, these DCNN models do not provide flexibility in accepting an input image with a given resolution (size). Rather, the input size was fixed at (224, 224, 3) in each of these DCNN models. The classification results were obtained from the individual models in terms of the accuracy, number of trainable parameters, and FLOPs. Next section contains a detailed analysis of the results.

IV. RESULT ANALYSIS AND DISCUSSION

Table 3 lists the numerical class-wise classification results for the proposed model.

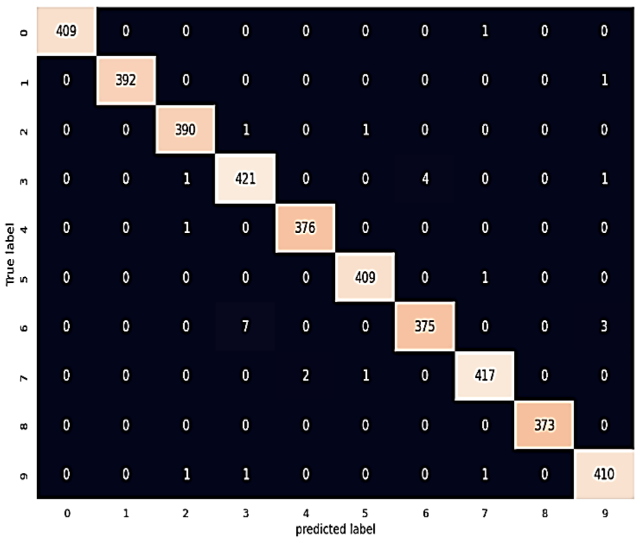
The proposed model achieved an overall top-1 recognition accuracy of 99.20% and top-5 recognition accuracy of 99.92%. The mean values of precision, recall, and F1-measure were 99.31%.

A confusion matrix was developed and is presented in Figure 8 to provide more insight into the results.

It can be observed that false predictions were minimal for most of the numeral classes. A little bit of struggle on the part of the model can be observed in the recognition of numeral classes 3 and 6. This might be due to their resemblance in shape to a horizontal flip (Fig. 1).

TABLE 2. List of performance metrics used to represent the numeral-class-wise classification results.

Metrics	Expression	Description
Recognition Accuracy (RA)	$\frac{TP+TN}{TP+FP+TN+FN}$	It counts the correct predictions out of total predictions.
Class - Precision (P_C)	$\frac{TP}{TP+FP}$	It counts the correct positive predictions out of the total positive predictions for given numeral class.
Mean - Precision (MP_C)	$Avg\left(\sum_{C=0}^{9} P_C\right)$	It counts the mean value of Class-Precision by considering all numeral classes; i.e., from class-0 to class-9.
Class - Recall (R_C)	$\frac{TP}{TP+FN}$	It counts the correct positive predictions out of the total samples for given numeral class.
Mean - Recall (MR_C)	$Avg\left(\sum_{C=0}^{9} R_C\right)$	It counts the mean value of Class-Recall by considering all numeral classes; i.e., from class-0 to class-9.
F1- measure ($F1_C$)	$2*\frac{P_C R_C}{P_C+R_C}$	It counts harmonic mean of Class-Precision and Recall-Precision for given class.
Mean- F1-measure ($MF1_C$)	$MF1_C = Avg\left(\sum_{C=0}^{9} F1_C\right)$	It counts mean F1-score considering all numeral classes i.e., from class-0 to class-9.



True \ Predicted	0	1	2	3	4	5	6	7	8	9
0	409	0	0	0	0	0	0	1	0	0
1	0	392	0	0	0	0	0	0	0	1
2	0	0	390	1	0	1	0	0	0	0
3	0	0	1	421	0	0	4	0	0	1
4	0	0	1	0	376	0	0	0	0	0
5	0	0	0	0	0	409	0	1	0	0
6	0	0	0	7	0	0	375	0	0	3
7	0	0	0	0	2	1	0	417	0	0
8	0	0	0	0	0	0	0	0	373	0
9	0	0	1	1	0	0	0	1	0	410

FIGURE 8. Confusion matrix related to the classification result of the proposed model.

TABLE 3. Numeral class-wise classification results of the proposed model.

Class	Class-Precision (% P_C)	Class-Recall (% R_C)	Class-F1-score (% $F1_C$)
0	100.00	99.80	99.90
1	100.00	99.70	99.90
2	99.20	99.50	99.40
3	97.90	98.60	98.20
4	99.50	99.70	99.60
5	99.50	99.80	99.60
6	98.90	97.40	98.20
7	99.30	99.30	99.30
8	100.00	100.00	100.00
9	98.80	99.30	99.00

TABLE 4. Results of pre-trained DCNN models (VGG-16Net, ResNet-50, and DenseNet-121).

Model	Overall recognition accuracy (% RA)	Parameter count (M)	FLOPS (G)
VGG-16Net	98.60	134.30	31.00
ResNet-50	98.60	23.60	7.75
DenseNet-121	93.26	7.04	5.70
Proposed model	99.20	0.218	0.0912

The model has a total of 0.218 M (actual value:217978) parameters, demonstrating the model's space demand. The computational complexity of the model is indicated by its ability to perform 0.0912 G floating-point operations per second. Table 4 summarizes the results obtained from the pretrained DCNN models (VGG-16Net, ResNet-50, and DensNet-121) for the same classification problem.

As shown in Table 4, the recognition accuracy of the proposed model is comparable to those of the DCNN models VGG-16Net and ResNet-50. The proposed model outperformed the other models in terms of parameter counts and FLOPs. The parameter counts have shown a clear difference; the corresponding values for VGG-16Net, ResNet-50, and DenseNet-121 were 134.30 M, 23.60 M, and 7.04 M, respectively, compared to the suggested model's 0.218 M. This demonstrates the ability of the proposed model to solve the given classification problem with a significantly lower space requirement. Further, the proposed model involved the least FLOPs as 0.0912 G against the 31.00 G, 7.75 G, and 5.70 G of the other models: VGG-16Net, ResNet-50, and DenseNet-121. This suggests the potential of the proposed model to solve the given problem with nominal computational complexity. These results validated the proposed model.

V. CONCLUSION

Machine-based recognition of handwritten Devanagari characters has always been tedious because of the richness of curves and degree of resemblance presented in the script. Several benchmarking models based on the DCNN handled the problem well and produced excellent results. The main concerns of these models are their large number of parameters (space requirements) and computational complexity. To address these concerns, the proposed model uses a customized shifted window transformer for the first time to solve the classification problem of handwritten Devanagari numerals. The proposed model exploits the capability of the SWIN transformer to handle information at given scales, which also has linear computational complexity concerning the input image. The model attained a comparable classification rate of 99.20 percent with the requirement of only 0.218 M parameters and 0.0912 G FLOPs. Popular pretrained DCNN models, VGG-16Net, ResNet-50, and DenseNet-121, were used to validate the results. The results of the proposed model suggest that the SWIN-transformer has the potential to replace convolutional neural networks in machine-based recognition of Devanagari numerals because of its capability to generate a comparable recognition rate with limited space and computational complexity requirements. The proposed model can have a significant impact in the fields of computer vision and natural language processing given the limitations of its computational resources.

ACKNOWLEDGMENT

The authors would like to thank the Google Co-Laboratory services to provide a hassle-free python platform with the power of a graphical processing unit and vast python library support, without which it would be not easy to complete the proposed work and also would like to thank Acharya, Pant, and Gyawali for their efforts in developing the dataset of handwritten Devnagari-characters and providing it in the public domain for progressive research in the related field.

REFERENCES

[1] R. Jayadevan, S. R. Kolhe, P. M. Patil, and U. Pal, "Offline recognition of devanagari script: A survey," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 41, no. 6, pp. 782–796, Nov. 2011.

[2] *Census of India Website: Office of the Registrar General & Census Commissioner, India*. Accessed: Dec. 22, 2020. [Online]. Available: https://censusindia.gov.in/2011Census/Language_MTs.html

[3] S. Arora, D. Bhattacharjee, M. Nasipuri, D. K. Basu, and M. Kundu, "Combining multiple feature extraction techniques for handwritten devnagari character recognition," in *Proc. IEEE Region 10 3rd Int. Conf. Ind. Inf. Syst.*, Dec. 2008, pp. 1–6, doi: 10.1109/ICIINFS.2008.4798415.

[4] S. R. Narang, M. K. Jindal, S. Ahuja, and M. Kumar, "On the recognition of Devanagari ancient handwritten characters using SIFT and Gabor features," *Soft Compsut.*, vol. 24, pp. 17279–17289, May 2020, doi: 10.1007/s00500-020-05018-z.

[5] N. Sharma, U. Pal, F. Kimura, and S. Pal, "Recognition of off-line handwritten Devnagari characters using quadratic classifier," in *Proc. ICVGIP*. Berlin, Germany: Springer-Verlag, 2006, pp. 805–816, doi: 10.1007/11949619_72.

[6] P. S. Deshpande, L. Malik, and S. Arora, "Handwritten Devnagari character recognition using connected segments and minimum edit distance," in *Proc. IEEE Region 10 Conf. (TENCON)*, Oct. 2007, pp. 1–4, doi: 10.1109/TENCON.2007.4428774.

[7] S. Shelke and S. Apte, "A fuzzy based classification scheme for unconstrained handwritten Devanagari character recognition," in *Proc. Int. Conf. Commun., Inf. Comput. Technol. (ICCICT)*, Jan. 2015, pp. 1–6, doi: 10.1109/ICCICT.2015.7045738.

[8] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, Jan. 2015, pp. 1135–1143.

[9] M. Mattina, "Co-designing hardware and models for efficient on-device ML inference," in *Proc. IEEE/ACM Int. Symp. Low Power Electron. Design (ISLPED)*, Boston, MA, USA, Jul. 2021, p. 1, doi: 10.1109/islped52811.2021.9502470.

[10] V. Sze, Y. Chen, J. Emer, A. Suleiman, and Z. Zhang, "Hardware for machine learning: Challenges and opportunities," in *Proc. IEEE Custom Integr. Circuits Conf. (CICC)*, Apr. 2018, pp. 1–8, doi: 10.1109/CICC.2018.8357072.

[11] P. Maji and R. Mullins, "On the reduction of computational complexity of deep convolutional neural networks," *Entropy*, vol. 20, no. 4, p. 305, Apr. 2018, doi: 10.3390/e20040305.

[12] V. Ashish, S. Noam, P. Niki, and U. Jakob, "Attention is all you need," in *Proc. Conf. Neural Inf. Process. Syst. (NIPS)*, no. 31, Long Beach, CA, USA, 2017, pp. 1–11. [Online]. Available: https://arxiv.org/abs/1706.03762

[13] R. Dai, C. Liu, and B. Xiao, "Chinese character recognition: History, status and prospects," *Frontiers Comput. Sci. China*, vol. 1, no. 2, pp. 126–136, May 2007, doi: 10.1007/s11704-007-0012-5.

[14] L. Yuan, Y. Chen, T. Wang, W. Yu, Y. Shi, Z. Jiang, F. E. H. Tay, J. Feng, and S. Yan, "Tokens-to-token ViT: Training vision transformers from scratch on ImageNet," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 538–547, doi: 10.1109/ICCV48922.2021.00060.

[15] S. Zheng, J. Lu, H. Zhao, X. Zhu, Z. Luo, Y. Wang, Y. Fu, J. Feng, T. Xiang, P. H. S. Torr, and L. Zhang, "Rethinking semantic segmentation from a sequence-to-sequence perspective with transformers," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 6877–6886, doi: 10.1109/CVPR46437.2021.00681.

[16] J. Yang, E. Chew, and P. Liu, "Service humanoid robotics: A novel interactive system based on bionic-companionship framework," *PeerJ Comput. Sci.*, vol. 7, pp. 1–20, Aug. 2021, doi: 10.7717/peerj-cs.674.

[17] R. Bajaj, L. Dey, and S. Chaudhury, "Devnagari numeral recognition by combining decision of multiple connectionist classifiers," *Sadhana*, vol. 27, no. 1, pp. 59–72, Feb. 2002.

[18] U. Bhattacharya, S. K. Parui, B. Shaw, and K. Bhattacharya, "Neural combination of ANN and HMM for handwritten Devanagari numeral recognition," in *Proc. 10th Int. Work. Frontiers Handwriting Recognit.*, La Baule, France, Oct. 2006, pp. 1–6.

[19] M. Hanmandlu and O. V. R. Murthy, "Fuzzy model based recognition of handwritten numerals," *Pattern Recognit.*, vol. 40, no. 6, pp. 1840–1854, Jun. 2007, doi: 10.1016/j.patcog.2006.08.014.

[20] C. V. Lakshmi, R. Jain, and C. Patvardhan, "Handwritten Devnagari numerals recognition with higher accuracy," in *Proc. Int. Conf. Comput. Intell. Multimedia Appl. (ICCIMA)*, Dec. 2007, pp. 255–259, doi: 10.1109/ICCIMA.2007.443.

[21] A. N. Holambe, S. Holambe, and R. C. Thool, "Comparative study of devanagari handwritten and printed character & numerals recognition using nearest-neighbor classifiers," in *Proc. 3rd Int. Conf. Comput. Sci. Inf. Technol. (ICCSIT)*, Jul. 2010, pp. 426–430, doi: 10.1109/ICCSIT.2010.5565024.

[22] N. Das, J. M. Reddy, R. Sarkar, S. Basu, M. Kundu, M. Nasipuri, and D. K. Basu, "A statistical–topological feature combination for recognition of handwritten numerals," *Appl. Soft Comput.*, vol. 12, no. 8, pp. 2486–2495, Aug. 2012, doi: 10.1016/j.asoc.2012.03.039.

[23] S. Iamsa-At and P. Horata, "Handwritten character recognition using histograms of oriented gradient features in deep learning of artificial neural network," in *Proc. Int. Conf. IT Converg. Secur. (ICITCS)*, Dec. 2013, pp. 1–5, doi: 10.1109/ICITCS.2013.6717840.

[24] P. Jarungthai, S. Chiewchanwattana, and K. Sunat, "Handwritten character recognition using generalized radial basis function extreme learning machine with centers selection," in *Proc. Asia–Pacific Signal Inf. Process. Assoc. Annu. Summit Conf. (APSIPA)*, Dec. 2014, pp. 1–5, doi: 10.1109/APSIPA.2014.7041773.

[25] D. Khanduja, N. Nain, and S. Panwar, "A hybrid feature extraction algorithm for Devanagari script," *ACM Trans. Asian Low-Resour. Lang. Inf. Process.*, vol. 15, no. 1, pp. 1–10, Jan. 2016, doi: 10.1145/2710018.

[26] S. Chaurasia and S. Agarwal, "Recognition of handwritten numerals of various Indian regional languages using deep learning," in *Proc. 5th IEEE Uttar Pradesh Sect. Int. Conf. Electr., Electron. Comput. Eng. (UPCON)*, Nov. 2018, pp. 1–6, doi: 10.1109/UPCON.2018.8596818.

[27] D. Rajpal and A. R. Garg, "Fusion-based feature extraction approach for recognition of handwritten Devanagari numerals," in *Proc. Int. Conf. Data Sci. Appl.*, in Lecture Notes in Networks and Systems, vol. 287. Singapore: Springer, 2021, pp. 1–13, doi: 10.1007/978-981-16-5348-3_12.

[28] A. R. Garg and D. Rajpal, "Offline handwritten devnagari character recognition using multilevel feature extraction and classification scheme," *Annu. Tech., Comput. Eng. Division Board, Inst. Eng. India*, vol. 3, pp. 102–109, 2020.

[29] S. Acharya, A. K. Pant, and P. K. Gyawali, "Deep learning based large scale handwritten Devanagari character recognition," in *Proc. 9th Int. Conf. Softw., Knowl., Inf. Manage. Appl. (SKIMA)*, Dec. 2015, pp. 1–6, doi: 10.1109/SKIMA.2015.7400041.

[30] B. Chakraborty, B. Shaw, J. Aich, U. Bhattacharya, and S. K. Parui, "Does deeper network lead to better accuracy: A case study on handwritten Devanagari characters," in *Proc. 13th IAPR Int. Workshop Document Anal. Syst. (DAS)*, Apr. 2018, pp. 411–416, doi: 10.1109/DAS.2018.72.

[31] M. Jangid and S. Srivastava, "Deep ConvNet with different stochastic optimizations for handwritten Devanagari character," in *Advances in Computer Communication and Computational Sciences*, vol. 759. Singapore: Springer, Mar. 2019.

[32] P. K. Sonawane and S. Shelke, "Handwritten Devanagari character classification using deep learning," in *Proc. Int. Conf. Inf., Commun., Eng. Technol. (ICICET)*, Aug. 2018, pp. 1–4, doi: 10.1109/ICICET.2018.8533703.

[33] N. Aneja and S. Aneja, "Transfer learning using CNN for handwritten Devanagari character recognition," in *Proc. 1st Int. Conf. Adv. Inf. Technol. (ICAIT)*, Jul. 2019, pp. 293–296, doi: 10.1109/ICAIT47043.2019.8987286.

[34] R. Guha, N. Das, M. Kundu, M. Nasipuri, and K. C. Santosh, "DevNet: An efficient CNN architecture for handwritten Devanagari character recognition," *Int. J. Pattern Recognit. Artif. Intell.*, vol. 34, no. 12, Nov. 2020, Art. no. 2052009, doi: 10.1142/S0218001420520096.

[35] G. S. Bhati and A. R. Garg, "Handwritten Devnagari character recognition using CNN with transfer learning," in *Proc. Congr. Intell. Syst., Adv. Intell.* Singapore: Springer, 2021, pp. 269–279.

[36] D. Rajpal, A. R. Garg, O. P. Mahela, H. H. Alhelou, and P. Siano, "A fusion-based hybrid-feature approach for recognition of unconstrained offline handwritten Hindi characters," *Future Internet*, vol. 13, no. 9, p. 239, Sep. 2021, doi: 10.3390/fi13090239.

[37] N. Alrobah and S. Albahli, "A hybrid deep model for recognizing Arabic handwritten characters," *IEEE Access*, vol. 9, pp. 87058–87069, 2021, doi: 10.1109/ACCESS.2021.3087647.

[38] A. Rasheed, N. Ali, B. Zafar, A. Shabbir, M. Sajid, and M. T. Mahmood, "Handwritten Urdu characters and digits recognition using transfer learning and augmentation with AlexNet," *IEEE Access*, vol. 10, pp. 102629–102645, 2022, doi: 10.1109/ACCESS.2022.3208959.

[39] Y. Li and Y. Li, "Design and implementation of handwritten Chinese character recognition method based on CNN and TensorFlow," in *Proc. IEEE Int. Conf. Artif. Intell. Comput. Appl. (ICAICA)*, Jun. 2021, pp. 878–882, doi: 10.1109/ICAICA52286.2021.9498146.

[40] S. Aly and S. Almotairi, "Deep convolutional self-organizing map network for robust handwritten digit recognition," *IEEE Access*, vol. 8, pp. 107035–107045, 2020, doi: 10.1109/ACCESS.2020.3000829.

[41] D. Gupta and S. Bag, "CNN-based multilingual handwritten numeral recognition: A fusion-free approach," *Expert Syst. Appl.*, vol. 165, Mar. 2021, Art. no. 113784, doi: 10.1016/j.eswa.2020.113784.

[42] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015, doi: 10.1038/nature14539.

[43] J. Beal, E. Kim, E. Tzeng, D. H. Park, A. Zhai, and D. Kislyuk, "Toward transformer-based object detection," 2020, *arXiv:2012.09958*.

[44] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 1–14, doi: 10.1109/ICCV48922.2021.00986.

[45] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. 3rd Int. Conf. Learn. Represent. (ICLR)*, 2015, pp. 1–14.

[46] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778, doi: 10.1109/CVPR.2016.90.

[47] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 2261–2269, doi: 10.1109/CVPR.2017.243.

DANVEER RAJPAL received the bachelor's degree (Hons.) in electronics and communication engineering from the University of Rajasthan, Jaipur, India, in 2004, and the master's degree (Hons.) in digital communication from JNU, Jodhpur, India, in 2011. He is currently pursuing the Ph.D. degree with the Department of Electrical Engineering, Faculty of Engineering and Architecture, J.N.V. University Jodhpur, India.

He has served as an Assistant Professor with the Department of Electronics and Communication, JECRC Engineering College, Jodhpur, from 2004 to 2012, and given his services as the Head of the Department of Electronics and Communication Engineering, VIET Engineering College, Jodhpur, from 2012 to 2019. His research interests include pattern recognition using machine learning and deep learning techniques.

AKHIL RANJAN GARG (Member, IEEE) received the B.E. degree in electrical engineering and the M.E. degree in control systems from the M. B. M. Engineering College, Jodhpur, India, and the Ph.D. degree from IIT Delhi, Delhi, India.

He is currently a Professor and the Head of the Department of Electrical Engineering, Faculty of Engineering and Architecture, MBM University, Jodhpur. His research interests include computational neuroscience, intelligent systems, pattern recognition, power electronics, and machine learning. He has published over 40 papers in these areas in refereed international and national journals and conference proceedings.

Dr. Garg is a recipient of numerous honors and awards, including the Young Teacher Career Award of AICTE, German Academic Exchange Program (DAAD) Fellowship, U.S. Naval Academy Fellowship, and International Brain Research Organization (IBRO) Fellowship. He is serving as a member for All India Board of Undergraduate Studies in Engineering and Technology (AIB-UGET) constituted by All India Council of Technical Education (AICTE), New Delhi. He is a member of the Board of Governor IIT Jodhpur, a member of the Executive Council Central University of Rajasthan, a member of the Board of Management, Rajasthan Technical University Kota, a member of the Academic Council MDS University Ajmer, and a member of the Academic Council, J. N. V. University, Jodhpur. He is former Honorary Chairperson, Institution of Engineers (India) Jodhpur Local Centre. He is a fellow of the Institution of Engineers (India), a Life Member of the Indian Society of technical Education (ISTE), a member of the International Brain Research Organization (IBRO), and a member of the International Neural Network Society (INNS).