IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier

Web-SpikeSegNet: deep learning framework for recognition and counting of spikes from visual images of wheat plants

TANUJ MISRA[1,5], ALKA ARORA[1], SUDEEP MARWAHA[1], RANJEET RANJAN JHA[2], MRINMOY RAY [1], A R RAO [1], ELDHO VARGHESE [4], SHAILENDRA KUMAR [5], SUDHIR KUMAR[3], ADITYA NIGAM [2], RABI NARAYAN SAHOO [3] AND VISWANATHAN CHINNUSAMY [3]

[1]Division of Computer Application, ICAR-Indian Agricultural Statistics Research Institute, New Delhi, India
[2]School of Computing andElectrical Engineering (SCEE),Indian Institute of Technology Mandi, India.
[3]ICAR-Indian Agricultural Research Institute, LibraryAvenue, New Delhi, India
[4]ICAR-Central MarineFisheries Research Institute, Kochi, India
[5]Rani Lakshmi Bai Central Agricultural University, Jhansi, India

Corresponding author: Alka Arora (e-mail: Alka.Arora@icar.gov.in).

The first author acknowledges the fellowship received from ICAR-IASRI, New Delhi, India to undertake this researchwork as part of Ph.D. Also acknowledges Nanaji Deshmukh Plant Phenomics Facility, ICAR-IARI, New Delhi-12, "This work was supported by National Agriculture Science Fund (NASF), ICAR, Grant No.NASF/Phen-6005/2016-17 and NAHEP CAAST (NAHEP/CAAST/2018/19/07)."

ABSTRACT Computer vision with deep-learning is emerging as a major approach for non-invasive and non-destructive plant phenotyping. Spikes are the reproductive organs of wheat plants. Detection and counting of spikes considered the grain-bearing organ have great importance in the phenomics study of large sets of germplasms. In the present study, we developed an online platform "Web-SpikeSegNet" based on a deep-learning framework for spike detection and counting from the wheat plant's visual images. The architecture of the Web-SpikeSegNet consists of 2 layers. First Layer, Client-Side Interface Layer, deals with end user's requests and corresponding responses management. In contrast, the second layer, Server Side Application Layer, consists of a spike detection and counting module. The backbone of the spike detection module comprises of deep encoder-decoder network with hourglass for spike segmentation. The Spike counting module implements the "Analyze Particle" function of imageJ to count the number of spikes. For evaluating the performance of Web-SpikeSegNet, we acquired the wheat plant's visual images, and the satisfactory segmentation performances were obtained as Type I error 0.00159, Type II error 0.0586, Accuracy 99.65%, Precision 99.59% and F_1 score 99.65%. As spike detection and counting in wheat phenotyping are closely related to the yield, Web-SpikeSegNet is a significant step forward in the field of wheat phenotyping and will be very useful to the researchers and students working in the domain.

INDEX TERMS Computer vision, deep learning, image analysis, spike detection and counting, Web-SpikeSegNet, wheat

I. INTRODUCTION

Wheat is one of the major food crops grown yearly on 215 million hectares globally [Wheat in the world CGIAR: *https://wheat.org/wheat -in-the-world/*]. It supersedes maize and rice in terms of protein sources in low- and middle-income nations. Climate change and associated abiotic stresses are the key factors of yield loss in wheat. Generic improvement in yield and climate resilience is critical for sustaining food security. One of the key aspects of genetic improvement is the determination of complex genome × environment × management interactions [1]. High-dimensional plant phenotyping is needed to bridge the genotype-phenotype gap in plant breeding and plant health monitoring in precision farming. Visual imaging is the most commonly used cost-effective method to quantitatively study of plant growth, yield, and adaptation of biotic and abiotic stresses. Besides, it is strongly reasoned that the imminent trend in plant phenotyping will depend on imaging sensors' combined tools and machine learning [2]. Yield estimation in wheat has received significant attention from researchers. The number of spikes/ears determines the grain number per unit area and thus yield. Counting of spikes of the large number of genotypes through traditional methods using naked-eye is a tedious and time-consuming job. Presently, non-destructive image analysis-based phenotyping is gaining momentum and proves as the less laborious and fast method. A cluster of research works available in the area of computer vision to detect and characterize spikes, and spikelets in wheat plants [3]–[8].High resolution image dataset with significant quantity is a major constraint to develop the computer vision based approaches. In this context, Pound et al. (2017) [6] and David et al. (2020) [9] contributed ACID (Annotated Crop Image Dataset) and GWHD (Global Wheat Head Detection) dataset respectively. In computer vision, the problem of spike detection lies under the domain of pixel-wise segmentation of objects. Bi et al. (2010) [4], Qiongyan et al. (2017) [5] and Sadeghi-Tehran et al. (2017) [7] used manually defined color intensities and textures for spike segmentation. Pound et al. (2017) [6] and Hasan et al. (2018) [8] used Autoencoder [10] and Region-based Convolutional Neural Network (R-CNN) [10] deep-learning technique, respectively, to detect and characterize spikes with greater than 90 percent accuracy. Xiong et al. (2019) [11] proposed a deep-learning model "TasselNetV2" to characterize the maize tassels with around 91% accuracy. Sadeghi-Tehran et al. (2019) [12]developed a methodology using Simple Linear Iterative Clustering and Deep Convolutional Neural Networks for the spike quantification in wheat plant. Recently, Misra et al.(2020) [3] developed a deep learning model known as SpikeSegNet, which was reported as an effective and robust approach for spike detection (accuracy: 99.91 percent) and counting (accuracy: 95 percent) from visual images irrespective of various illumination factors. In this paper, a web-solution is presented as "Web-SpikeSegNet" for spike segmentation and counting from wheat plants' visual images for easy accessibility and quick reference. The developed web-solution has a wide application in the plant phenomics domain and will be useful for researchers and students working in the field of wheat plant phenotyping. Web-SpikeSegNet is platform-independent and is readily accessible by at the URL: http://spikesegnet.iasri.res.in/.

II. IMPLEMENTATION

Web_SpikeSegNet is developed based on the approach give by Misra et al. (2020) [3]. The approach is based on the convolutional encoder-decoder deep-learning technique for pixel-wise segmentation of spikes from the wheat plant's visual images. The architecture of the network was inspired by UNet [13], SegNet [14], and PixISegNet [15], which are popularly used in various sectors for pixel-wise segmentation of objects. SpikeSegNet consists of two modules *viz.,* Local Patch extraction Network (LPNet) and Global Mask Refinement Network (GMRNet), in sequential order. The details of the approach are given in [3]. Input images were divided into patches before entering into the LPNet module to facilitate local features' learning more effectively than the whole input image. LPNet was used in extracting and understanding the contextual and local features at the patch level. Output images of the LPNet are further refined at GMRNet to better segment the spikes, as given in Figure 1. SpikeSegNet network was trained using visual images of the wheat plant and its corresponding ground-truth segmented mask images with class labels (*i.e.,* spike regions of the plant image). Details of the dataset preparation for training the network were given in [3]. SpikeSegNet provides significant segmentation performance at pixel-level in spike detection and counting and is also proved as a robust approach when tested for different illumination levels that may occur in the field conditions.

A. ARCHITECTURE OF THE PROPOSED SOFTWARE — "WEB-SPIKESEGNET"

Web-SpikeSegNet is web-based software for the detection and counting of spikes from visual images of the wheat plant. It is developed and implemented on the Linux operating system with 32 GB RAM and NVIDIA GeForce GTX 1080 Ti graphics card (with a memory of 11 GB). PyCharm version 5.0 integrative development environment developed by Jetbrains [*https://www.jetbrains.com/*] was used for the development of the software. The software architecture consists of two layers, namely Client-Side Interface Layer (CSIL) and Server Side Application Layer (SSAL). The architecture of Web-SpikeSegNet is given in Fig. 2. End-users (especially the plant physiologist) will interact with the Web-SpikeSegNet available at *http://spikesegnet.iasri.res.in/* through CSIL using internet. CSIL deals with the end-user's requests and its corresponding responses management and implemented using HyperText Markup Language (HTML) [16], Cascading Style Sheets (CSS) [17], Flask [18], and JavaScript [19] technologies. HTML, CSS and Flask were used to design the front-end view of the webpages and JavaScript was used for the client side validation. End-users will upload wheat imege in the software through CSIL and then it will be forwarded to the SSAL for the spike detection and counting. SSAL consists of two modules: spike detection and spike counting module. SpikeSegNet deep learning model will be applied on the input image for the spike segmentation in Spike Detection module and it will be forwarded to the spike counting module for counting the segmented spikes. After completion of the process, the segmented spikes along with spike count will be shown in the end-user's window through CSIL. Spike detection module was developed using python libraries such as Tensorflow [20], Keras [21], Numpy [22], Scipy [23], Matplotlib [24] and OpenCV [25] for constructing and implementing the deep learning model.Convolutional encoder network [10] (Encoder_SpikeSegNet), decoder network [10] (Decoder_SpikeSegNet), and bottleneck network ([10], [15]) using stacked hourglasses (Bottleneck_SpikeSegNet) are the backbone of LPNet, GMRNet and correspondingly

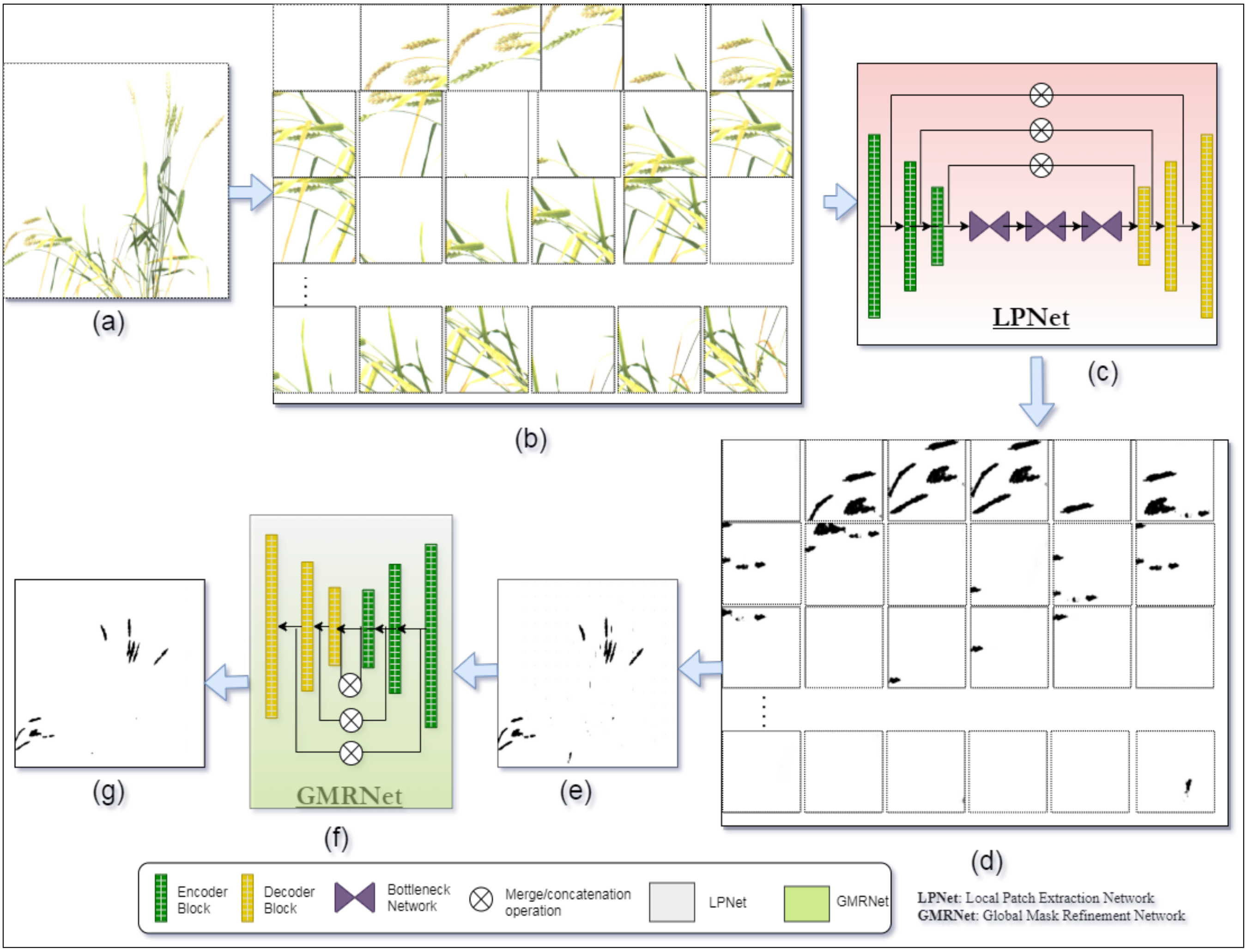

FIGURE 1. Flow diagram of SpikeSegNet: Here, input is visual image of wheat plant of size 1656*1356 . The input image is divided into patches of size 256*256 before entering into the LPNet. The output of LPNet are patch-by-patch segmented mask images which are then combined to form the mask image as per the size of the input visual image. This image may contain some sort of inaccurate segmentation of the object (or, spikes) and are refined at global level using GMRNet network.The output of GMRNet network is nothing but the refined mask image containing spike regions only.

the SpikeSegNet.The number of encoders, decoders, and stacked hourglasses was estimated empirically, as given in [3], to produce the best results by considering the optimum performances. Encoder_SpikeSegNet consists of 3 encoder blocks, and the output feature-maps of each encoder block are forwarded to the next encoder block for further feature extraction. Each encoder block consists of two convolution layers, each with the square filter of size 3*3 [26] with a varying number of filters (16, 64, 128) followed by ReLU [27] and max-pooling layer with a window size of 2*2 [28]. Square filters are popularly used in state-of-art methods [29], and the mentioned window size is considered as standard [13], [30]. Batch Normalization, a statistical procedure, is done to improve the performance as well as stability of the network. Input and output feature description of each encoder block in the Encoder_SpikeSegNet is presented in the tabular form (Table 1) and the algorithm for implementing the Encoder_SpikeSegNet network is given in Algorithm 1.

Decoder_SpikeSegNet network facilitates a special operation called transpose convolution [31], which up-sampled the incoming features to regenerate or decode the same. The resulting up-sampled feature maps are then concatenated/ merged with the corresponding encoded feature maps of the Encoder_SpikeSegNet. Merge operation helps in transferring the spatial information across the network for better localization of the segmented masks. The Decoder_SpikeSegNet contains three decoder blocks, and each decoder block consists of two convolution layers (with filter size 3*3) with a varying number of filters (128, 64, 16) as opposite to each encoder block in Encoder_SpikeSegNet and followed by ReLU operation to decode the features. The output of the final decoder was fed into the “SoftMax” ([32]) activation layer for classifying objects (or spikes).Input and output feature description of each decoder block in the Decoder_SpikeSegNet is presented in the tabular form (Table 2) and the algorithm for implementing the Decoder_SpikeSegNet network

TABLE 1. Input and output feature description of each encoder block in the Encoder_SpikeSegNet Network

Encoder Block #	Name of the Layers	Input feature size	# of kernel with size 3*3	Output feature size
Encoder Block-1	E_conv_1_1^p	256*256*1	16	256*256*16
	E_conv_1_2^p	256*256*16	16	256*256*16
	Pool-1	256*256*16	-	128*128*16
Encoder Block-2	E_conv_2_1^p	128*128*16	64	128*128*64
	E_conv_2_2^p	128*128*64	64	128*128*64
	Pool-2	128*128*64	-	64*64*64
Encoder Block-3	E_conv_3_1^p	64*64*64	128	64*64*128
	E_conv_3_2^p	64*64*128	128	64*64*128
	Pool-3	64*64*128	-	32*32*128

^{p}Each convolution layer is followed by ReLU activation function and batch normalization
Feature size=x*y*z represents z number of features with x*y size
E_conv_u_v denotes the vth convolution layer of the uth encoder block number

TABLE 2. Input and output feature description of each decoder block in the Decoder_SpikeSegNet Network

Decoder Block #	Name of the Layers	Input feature size	# of kernel with size 3*3	Output feature size
Decoder Block-1	T_conv-1^p	32*32*128	128	64*64*128
	D_conv_1_1^q	64*64*128	128	64*64*128
	D_conv_1_2^q	64*64*128	128	64*64*128
Decoder Block-2	T_conv-2^p	64*64*128	64	128*128*64
	D_conv_2_1^q	128*128*64	64	128*128*64
	D_conv_2_2^q	128*128*64	64	128*128*64
Decoder Block-3	T_conv-3^p	128*128*64	16	256*256*16
	D_conv_3_1^q	256*256*16	16	256*256*16
	D_conv_3_2^q	256*256*16	16	256*256*16

^{p}Transpose convolution operation followed by batch normalization and merge operation with the corresponding encoder block output
^{q}Convolution operation followed by batch normalization

is given in Algorithm 2.Bottleneck_SpikeSegNet network contains three hourglasses, which provide more confident segmentation by concentrating the essential features captured at various occlusions, scale, and view-points [8], [13]. Each hourglass comprises a sequence of residual blocks containing three convolution layers of filter size 1*1, 3*3, and 1*1 sequentially with depth (or the number of filters) 128, 128, and 256, respectively, estimated empirically on the basis of optimal performances. Algorithms for implementing Bottleneck _SpikeSegNet, LPNet, and GMRNet are presented in Algorithm 3, 4, and 5, respectively. The Spike counting module is integrated with the output of the Spike detection module in SSAL. For this purpose, the "Analyze Particle" functions of imageJ [33] was applied to the output image of GMRNet, which is a segmented mask image or binary image containing spike region only. "Analyze Particle" function implements a flood-fill technique [34] for counting of object.

B. TRAINING OF WEB-SPIKESEGNET

For training the spike-detection module of Web-SpikeSegNet using the algorithms [1-5], 600 wheat plant's visual images were captured using LemnaTec imaging facility installed at Nanaji Deshmukh Plant Phenomics Center, New Delhi, India. The image dataset was randomly divided into training and testing at 85% and 15% respectively. Web-SpikeSegNet was trained for 300 epochs with batch size 32 due to the system platform constraints. Binary Cross-entropy loss function was used as it is a binary classification problem (i.e., pixels with either spike pixels or non-spike pixels) in the domain of image segmentation. Details of the hyper-parameters used to train the network are given in Table 3.

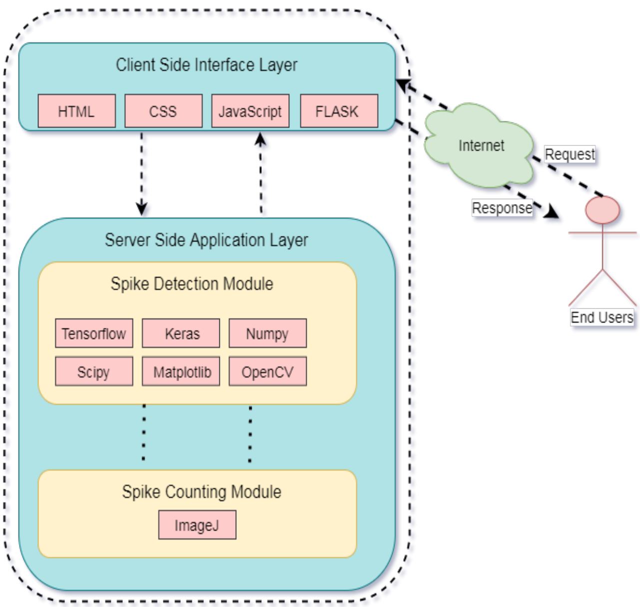

FIGURE 2. Architecture of Web-SpikeSegNet: The software architecture consists of two layers, namely Client-Side Interface Layer (CSIL) and Server Side Application Layer (SSAL). CSIL deals with the end-user's requests and its corresponding responses management. SSAL consists of two modules: spike detection and spike counting module.

Algorithm 1 Encode_SpikeSegNet: Encoding operation of SpikeSegNet

```
1: I: Input image/feature
2: Conv(input feature, filter_size, no. of filters): Convolution operation    ▷ for generating feature maps
3: BatchNorm(): Batch normalization operation    ▷ for improving the performance as well as stability of the network
4: Pool(): Pooling operation or down-sampling with window size 2*2
5: procedure ENCODER_SPIKESEGNET(I)    ▷ input image of size 256*256
6:     //First Encoder Block
7:     E_conv_1_1 ← Conv(I, 3*3, 16)    ▷ generates 16 feature maps of size 256*256
8:     E_batch_1_1 ← BatchNorm(E_conv_1 _1)    ▷ batch normalization of the features
9:     E_conv_1_2 ← Conv(E_batch_1 _1, 3*3, 16)    ▷ generates 16 feature maps from the batch normalized features
10:    E_batch_1_2 ← BatchNorm(E_conv_1_2)
11:    I_Encoded_block_1 ← Pool(E_batch_1_2)    ▷ size of each feature map reduced by half and returns 16 feature maps of size 128*128
12:    //Second Encoder Block. Here input is the output of First encoder block
13:    E_conv_2_1 ← Conv(I_Encoded_block _1, 3*3, 64)    ▷ generates 64 feature maps of size 128*128
14:    E_batch_2_1 ← BatchNorm(E_conv_2 _1)    ▷ batch normalization of the features
15:    E_conv_2_2 ← Conv(E_batch_2 _1, 3*3, 64)
16:    E_batch_2_2 ← BatchNorm(E_conv_2 _2)
17:    I_Encoded_block_2 ← Pool(E_batch_2 _2)    ▷ return 64 feature maps of size 64*64
18:    //Third Encoder Block. Here input is the output of second encoder block
19:    E_conv_3_1 ← Conv(I_Encoded_block _2, 3*3, 128)    ▷ generates 128 feature maps of size 64*64
20:    E_batch_3_1 ← BatchNorm(E_conv_3 _1)
21:    E_conv_3_2 ← Conv(E_batch_3 _1, 3*3, 128)
22:    E_batch_3_2 ← BatchNorm(E_conv_3 _2)
23:    I_Encoded_block_3 ← Pool(E_batch_3 _2)    ▷ return 128 feature maps of size 32*32
24: return I_Encoded_block _3
```

Algorithm 2 Decoder_SpikeSegNet: Decoding operation of SpikeSegNet

```
1: I: Output of Bottleneck_SpikeSegNet (for LPNet) or, output of Encoder_SpikeSegNet (for GMRNet).
2: Conv(input feature, filter_size, no. of filters): Convolution operation
3: BatchNorm(): Batch normalization operation
4: Tr_conv(input feature, filter_size, no. of filters): Transpose convolution    ▷ to up-sample the feature maps
5: Merge(): Merge/concatenation operation    ▷ for transferring the spatial information across the network
6: procedure DECODER_SPIKESEGNET(I)    ▷ here input is 128 feature maps of size 32*32
7:     //First Decoder Block
8:     T_conv_1 ← Tr_Conv(I, 3*3, 128)    ▷ Up-sampling done and return 128 decoded feature maps of size 64*64
9:     D_batch_1_1 ← BatchNorm(T_conv_1)    ▷ batch normalization of the features
10:    M_1 ← Merge(D_batch_1 _1, I_Encoded_block _3) ▷ concatenation operation with the output of third Encoder block [refer Algorithm 1 Line no.: 23]
11:    D_conv_1_1 ← Conv(M_1, 3*3, 128)
12:    D_batch_1_2 ← BatchNorm(D_conv_1 _1)    ▷ batch normalization of the features
13:    D_conv_1_2 ← Conv(D_batch_1 _1, 3*3, 128)
14:    I_Decoded_block_1 ← BatchNorm(D_conv_1 _2)    ▷ Output of the 1st Decoder Block is 128 decoded feature maps of size 64*64
15:    //Second Decoder Block. Here input is the output of First Decoder block
16:    T_conv_2 ← Tr_Conv(I_Decoded_block _1, 3*3, 64)    ▷ Up-sampling done and return 64 decoded feature maps of size 128*128
17:    D_batch_2_1 ← BatchNorm(T_conv_2)    ▷ batch normalization of the features
18:    M_2 ← Merge(D_batch_2 _1, I_Encoded_block _2)    ▷ concatenation operation with the output of second Encoder block [refer Algorithm 1 Line no.: 17]
19:    D_conv_2_1 ← Conv(M_2, 3*3, 64)
20:    D_batch_2_2 ← BatchNorm(D_conv_2 _1)
21:    D_conv_2_2 ← Conv(D_batch_2 _2, 3*3, 64)
22:    I_Decoded_block_2 ← BatchNorm(D_conv_2 _2)    ▷ Output of the second Decoder Block is 64 decoded feature maps of size 128*128
23:    //Third Decoder Block. Here input is the output of Second Decoder block
24:    T_conv_3 ← Tr_Conv(I_Decoded_block _2, 3*3, 16)    ▷ Up-sampling done and return 16 decoded feature maps of size 256*256
25:    D_batch_3_1 ← BatchNorm(T_conv_3)    ▷ batch normalization of the features
26:    M_3 ← Merge(D_batch_3 _1, I_Encoded_block _1) ▷ concatenation operation with the output of First Encoder block [refer Algorithm 1 Line no.: 11]
27:    D_conv_3_1 ← Conv(M_3, 3*3, 16)
28:    D_batch_3_2 ← BatchNorm(D_conv_3 _1)
29:    D_conv_3_2 ← Conv(D_batch_3 _2, 3*3, 16)
30:    I_Decoded_block_3 ← BatchNorm(D_conv_3 _2)    ▷ Output of the third Decoder Block is 16 decoded feature maps of size 256*256
31: return I_Decoded_block _3
```

C. PERFORMANCE MEASUREMENT OF WEB-SPIKESEGNET

For evaluating the segmentation performance to detect the spikes, the resulting segmented images (I^{pred}) using the Web-SpikeSegNet software are compared with the corresponding ground-truth mask images (I^{grtr}), which were prepared by ensuing the steps mentioned in [3]. Segmentation performances are calculated using the following [Eq. (1) to Eq. (10)] statistical parameters [35]–[37]:

Type I Error (E_1): For any r^{th} test image, exclusive-OR operation is done to compute pixel-wise classification error (Pix_Err_{r}) between (I^{pred}) and the corresponding (I^{grtr})

Algorithm 3 Bottleneck_SpikeSegNet

```
I: Input image/feature
Conv(input feature, filter_size, no. of filters): Convolution operation
BatchNorm(): Batch normalization operation
Tr_conv(input feature, filter_size, no. of filters): Transpose convolution operation
Pool(): Pooling operation or down-sampling with window size 2*2
Merge(): Merge/concatenation operation
procedure BOTTLENECK_SPIKESEGNET(I)        ▷ here, input is output of ENCODER_SPIKESEGNET, 128 feature maps of size 32*32
    H_1 ← HOURGLASS_SPIKESEGNET(I)        ▷ Call HOURGLASS_SPIKESEGNET procedure and return, 128 feature maps of size 32*32
    Scale_up_ ← SCALE_UP(H_1)        ▷ Call SCALE_UP procedure and return, 128 feature maps of size 64*64
    H_2 ← HOURGLASS_SPIKESEGNET(Scale_up)
    Scale_down_ ← SCALE_DOWN(H_2)        ▷ Call SCALE_DOWN procedure and return, 128 feature maps of size 32*32
    H_3_ ← HOURGLASS_SPIKESEGNET(Scale_down)
return H_3        ▷ return, 128 refined feature maps of size 32*32
        ▷ Hourglass gives more confident segmentation by concentrating on the essential features
procedure HOURGLASS_SPIKESEGNET(I)
    res_1 ← RESIDUAL_BL(I)        ▷ returns, 256 feature maps of size 32*32
    pool_1 ←Pool(res_1)        ▷ down-sampling done and returns, 256 feature maps of size 16*16
    res_2 ← RESIDUAL_BL(pool_1)        ▷ returns, 256 feature maps of size 16*16
    pool_2 ←Pool(res_2)        ▷ down-sampling done and returns, 256 feature maps of size 8*8
    res_3 ← RESIDUAL_BL(pool_2)        ▷ returns, 256 feature maps of size 8*8
    pool_3 ←Pool(res_3)        ▷ down-sampling done and returns, 256 feature maps of size 4*4
    res_4 ← RESIDUAL_BL(pool_3)        ▷ returns, 256 feature maps of size 4*4
    res_5 ← RESIDUAL_BL(res_4)
    T_conv_1 ←Tr_conv(res_5, 3 * 3, 256)        ▷ up-sampling done and returns, 256 feature maps of size 8*8
    M_1 ←Merge(T_conv_1, res_3)
    res_6 ← RESIDUAL_BL(M_1)        ▷ returns, 256 feature maps of size 8*8
    T_conv_2 ←Tr_conv(res_6, 3 * 3, 256)        ▷ up-sampling done and returns, 256 feature maps of size 16*16
    M_2 ←Merge(T_conv_2, res_2)
    res_7 ← RESIDUAL_BL(M_2)        ▷ returns, 256 feature maps of size 16*16
    T_conv_3 ←Tr_conv(res_7, 3 * 3, 256)        ▷ up-sampling done and returns, 256 feature maps of size 32*32
    M_3 ←Merge(T_conv_3, res_1)
    res_8 ← RESIDUAL_BL(M_3)        ▷ returns, 256 feature maps of size 32*32
return res_8
procedure RESIDUAL_BL(I)
    res_conv_1 ←Conv(I, 1 * 1, 128)
    res_conv_2 ←Conv(res_conv_1, 3 * 3, 128)
    res_conv_3 ←Conv(res_conv_2, 1 * 1, 256)
return res_conv_3        ▷ returns, 256 feature maps        ▷ Scale up and scale down operations help in finding the relationships among aggregate features at
different scales which further helps in getting the robust features
procedure SCALE_UP(I)
    sc_up_conv_1 ←Conv(I, 3 * 3, 128)
    sc_up_batch_1 ←BatchNorm(sc_up_conv_1)
    sc_up_conv_2 ←Conv(sc_up_batch_1, 3 * 3, 128)
    sc_up_batch_2 ←BatchNorm(sc_up_conv_2)
    sc_up_pool ←Tr _Pool(sc_up_batch_2)
return sc_up_pool
procedure SCALE_DOWN(I)
    sc_down_pool_1 ←Pool(I)
    sc_down_conv_1 ←BatchNorm(sc_down_pool_1, 3 * 3, 128 )
    sc_down_batch_1 ←BatchNorm(sc_down_conv_1)
    sc_down_conv_2 ←Conv(sc_down_batch_1, 3 * 3, 128)
    sc_down_batch_2 ←BatchNorm(sc_down_conv_2)
return sc_down_batch_2
```

Algorithm 4 LPNet Local Patch Extraction Network

```
I: Input image/feature
procedure LPNET(I)        ▷ here input is visual image patches of size 256*256
    Encoded_I← ENCODER_SPIKESEGNET(I)        ▷ Call Algorithm 1. Return encoded feature maps of the input image
    Bottleneck_I← BOTTLENECK_SPIKESEGNET(Encoded_I)        ▷ Call Algorithm 3. Return refined feature maps of the input features
    Decoded_I← DECODER_SPIKESEGNET(Bottleneck_I)        ▷ Call Algorithm 2. Return decoded feature maps of the input features
return Decoded_I        ▷ Segmeted mask image of size 256*256 containing spikes regions corresponding to the input patches.
```

Algorithm 5 GMRNet

```
I: Input image/feature
procedure GMRNET(I)        ▷ here input is the output image/feature of LPNet
    Encoded_I← ENCODER_SPIKESEGNET(I)        ▷ Call Algorithm 1. Return encoded feature maps of the input image
    Decoded_I← DECODER_SPIKESEGNET(Encoded_I)        ▷ Call Algorithm 2. Return decoded feature maps of the input features
return Decoded_I        ▷ Refined segmeted mask image of size 256*256 containing spikes regions corresponding to the input image/feature.
```

TABLE 3. Hyper-parameters

Optimizer	:	Adam
Learning rate	:	0.0005
Epoch	:	300
Batch size	:	32
Loss function	:	Binary Cross Entropy

image of size p×q,

$$Pix\_Err_{\mathrm{r}}(I^{\mathrm{pred}}, I^{\mathrm{grtr}}) = \frac{1}{p*q}\sum_{l=1}^{q}\sum_{k=1}^{p}[I^{\mathrm{pred}}(k,l) \oplus I^{\mathrm{grtr}}(k,l)] \quad (1)$$

E_1 is computed by averaging the Pix_Err_{r} of all the test images:

$$E_1 = \frac{1}{n}\sum_{r=1}^{n} Pix\_Err_{\mathrm{r}} \quad (2)$$

Where, n is the total number of test images. E_1 lies within [0, 1]. If the value of E_1 is close to "0", it refers minimum error, whereas if E_1 is close to "1", it signifies large error.

Type II error (E_2): For any r^{th} test image, the error rate $E_2{}^{\mathrm{r}}$ is computed by the average of false-positives (FPR) and false negatives (FNR) rates at the pixel level defined as:

$$E_2^r = 0.5 * FPR + 0.5 * FNR \quad (3)$$

Where,

$$FPR = \frac{1}{p*q}\sum_{l=1}^{q}\sum_{k=1}^{p}[(I^{\mathrm{grtr}}(k,l) .* I^{\mathrm{pred}}(k,l)) \oplus I^{\mathrm{pred}}(k,l)] \quad (4)$$

$$FNR = \frac{1}{p*q}\sum_{l=1}^{q}\sum_{k=1}^{p}[(I^{\mathrm{grtr}}(k,l) .* I^{\mathrm{grtr}}(k,l)) \oplus I^{\mathrm{pred}}(k,l)] \quad (5)$$

E_2 is computed by taking the average errors of all the input test images as given below:

$$E_2 = \frac{1}{n}\sum_{r=1}^{n} E_2^r \quad (6)$$

Following performance parameters are also used for measuring the segmentation performance of the Web-SpikeSegNet at pixel level to identify/detect spikes as follows:

- True positive (TP): number of pixels correctly classified as spikes.
- True Negative (TN): number of pixels correctly classified as non-spikes (other than spike pixels).
- False Positive (FP): number of non-spike pixels classified as spikes pixels.
- False Negative (FN): number of spike pixels classified as non- spikes pixels.

Then Precision, Recall, F-measure and Accuracy can be defined as:

$$Precision = TP/(TP + FP) \quad (7)$$

measures the percentage of detected pixels are actually spikes

$$Recall = TP/(TP + FN) \quad (8)$$

measures the percentage of actually spikes spike pixels are detected

$$Accuracy = (TP + TN)/(TP + TN + FP + FN) \quad (9)$$

measures performance of the Web-SpikeSegNet

$$F_1Score = 2(Precision * Recall)/(Precision + Recall) \quad (10)$$

measures robustness of the Web-SpikeSegNet in detecting or identifying spikes

III. RESULTS AND DISCUSSION

To demonstrate the working environment of Web-SpikeSegNet, a case study is presented here. The architecture of Web-SpikeSegNet mentioned in section 3, and the design of the software consists of 5 sections, namely "Home page", "Spike Detection and Counting", "Help", "Contact Us", and "Sample Data set". The "Home page" contains basic information about SpikeSegNet, and the flow diagram of the steps needs to be followed to recognize and count the spikes of the uploaded wheat plant image (Fig. 3). The "Sample Data set" section facilitates sample visual images of wheat plants for the experiment. Spike Detection and Counting module is the center of attention of the software. The user has to follow the following steps to detect and count the spikes and the output of each steps are pictorially presented in Supplementary 1:

1) Select and upload visual image of wheat plant of size 1656*1356 consisting of above ground parts only as discussed in [3].
2) Click on "Generate Patches" button for dividing the whole image into patches. Here, the visual image is divided into 100 pixel overlapping patches (each patches of size 256*256) which work as input to the LPNet module. Therefore, from one visual image of size 1656*1356, 180 patches of size 256*256 will be generated.
3) Click on "Run LPNet" to run the LPNet module for extracting contextual and spatial features at patch level. Output of the LPNet are the segmented images of size 256*256 corresponding to the patch images.
4) The output of LPNet are merged to generate the segmented image of size 1656*1656 that contains some inaccurate segmentation of spikes and further refined at global level by clicking on "Run GMRNet" button.
5) For counting the wheat spikes, click on "Count"button and the corresponding spikes count will be displayed on the next window.

The final output of Web-SpikeSegNet after detection and counting of spikes from the visual images of wheat plant is given in Fig. 4.

TABLE 4. Segmentation performance analysis of Web-SpikeSegNet

Type I Error	Type II Error	Accuracy	Precision	Recall	F1 Score
0.00159	0.0586	0.9965	0.9959	0.9961	0.9965

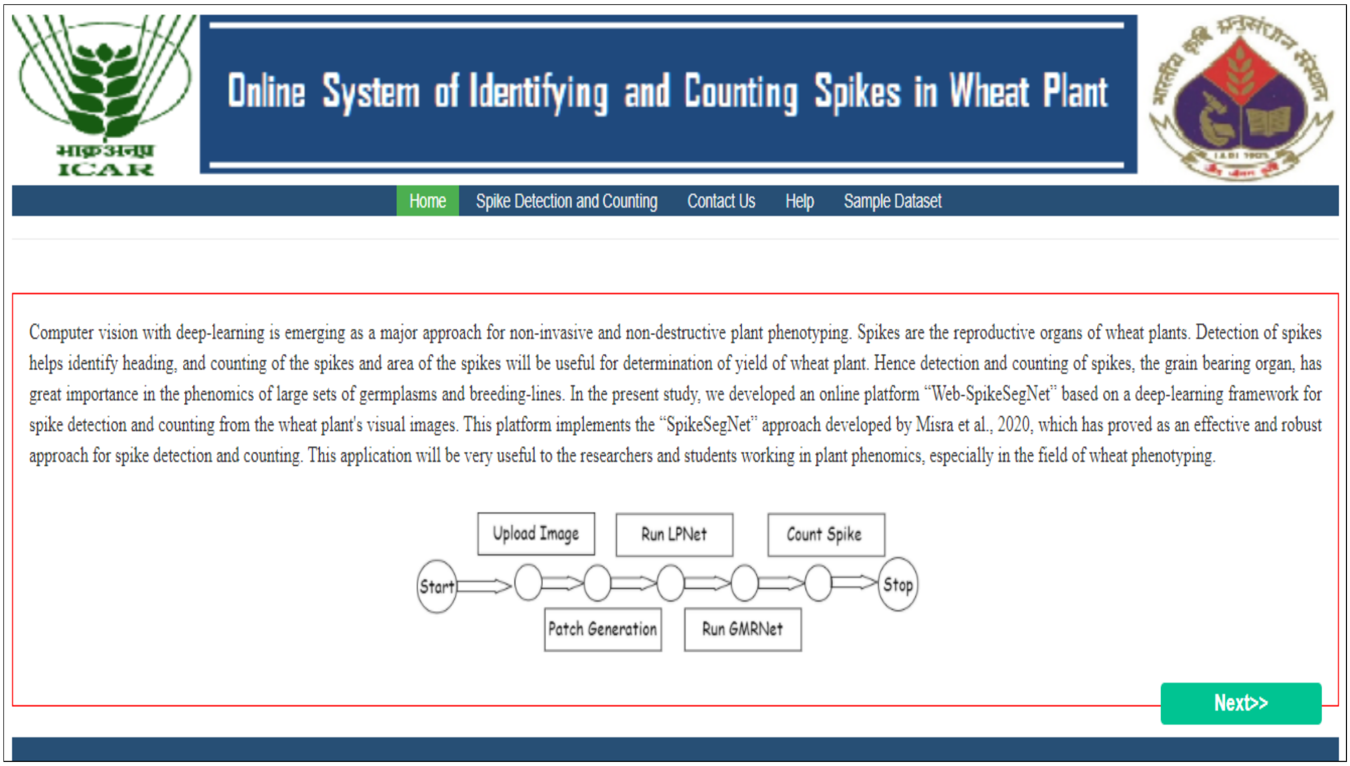

FIGURE 3. Home page of Web-SpikeSegNet contains basic information about SpikeSegNet and the flow diagram of the steps need to be followed to recognize and counting the spikes of the uploaded wheat plant image.

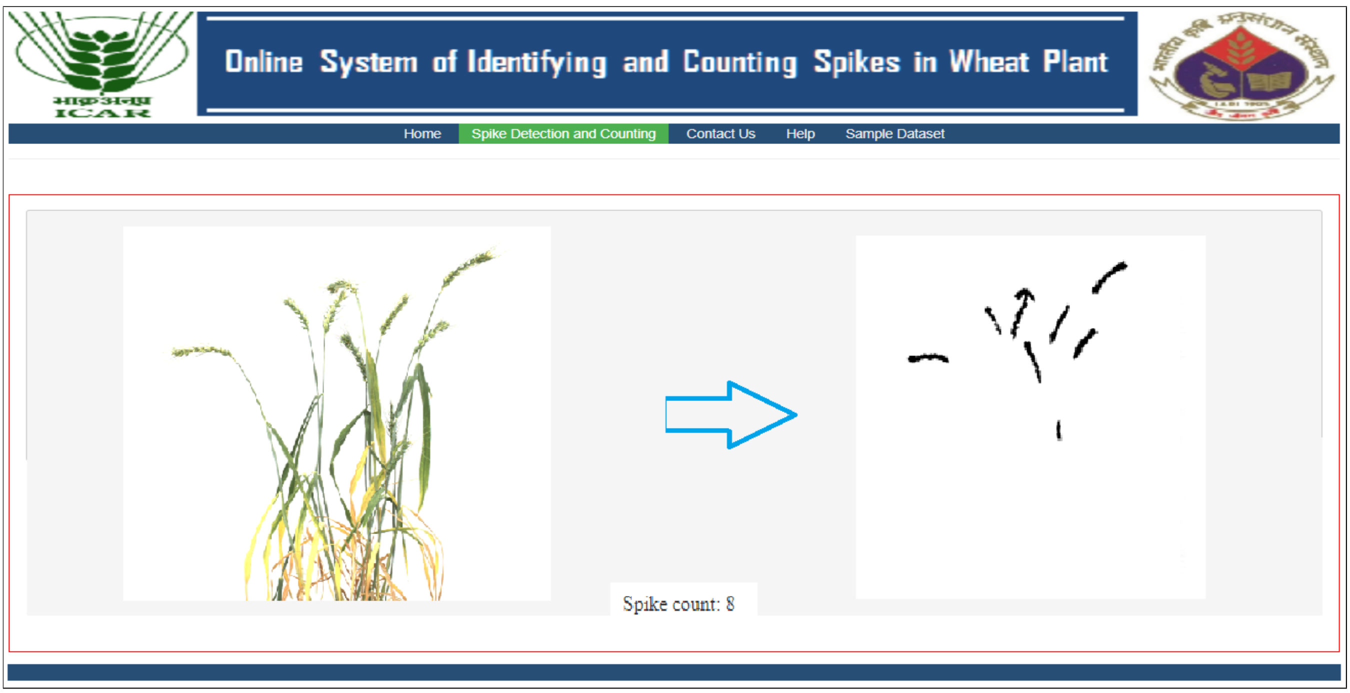

FIGURE 4. The final output of Web-SpikeSegNet after detection and counting of spikes from the visual images of wheat plant.

A. PERFORMANCE ANALYSIS OF WEB-SPIKESEGNET

Web-SPikeSegNet was trained using the training dataset consisting of randomly selected 85% of the total images captured (i.e., 510 images among 600 images). Although the network was trained for 300 epochs, the training losses were plateaued around 100 epoch as given in Fig 5. Segmentation performances of the Web-SpikeSegNet has been computed on the testing dataset consists of 90 images. The mentioned statistical parameters (eq. 1 to eq. 10) are computed, and the average values are presented in Table 4.As the performance of spike detection is calculated at the pixel level, the value of E1 (=0.00159) depict that on an average only 104 pixels are misclassified among 65,536 pixels which is the pixel size of one image, i.e., 65,536 (256 * 256). The accuracy of the approach as well as the developed software is around 99.65 %. The average precision value reflects that 99.59% of the detected spikes are actually spike pixels and the robustness of the approach is also $\sim 100\%$.

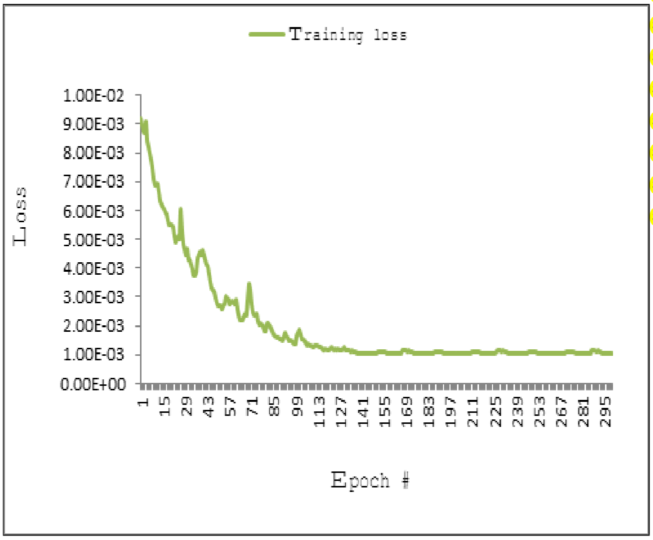

FIGURE 5. Graphical representation of training Loss.

B. COMPARATIVE ANALYSIS WITH ACID (ANNOTATED CROP IMAGE DATASET) DATASET AVAILABLE AT HTTPS://PLANTIMAGES.NOTTINGHAM.AC.UK/

For the comparative study, we ran the developed software on the ACID (Annotated Crop Image Dataset) dataset. The dataset consists of 415 training images and 105 testing images and was contributed by Pound et al (2017) [6]. They proposed a multi-task deep learning architecture for localizing wheat spikes and spikelet and achieved 95 % accuracy in spike detection. As the Web-SpikeSegNet model was trained using the wheat's visual images with consistent white background, we converted the background of the test images given in the mentioned website from back to white for conducting the comparative study. The output of Web-SpikeSegNet on ACID dataset is presented in Fig. 6. To compute the segmentation performance the ground-truth mask images corresponding to the testing dataset were prepared utilizing the procedure mentioned in [3]. The average segmentation performances are given in Table 5.The value of the type I error (0.00164) reflects that on an average only 107 pixels are wrongly classified among 65,536 pixels (size of one image is 256*256 pixels). The accuracy (99.55%), precision (99.62%), and F1 value (99.62%) depicts that Web-SpikeSegNet approach is comparatively generalized and robust than the approach presented by Pound et al (2017) [6]. It is due to the training criteria of Web-SpikeSegNet where, the deep learning model is trained at patch level for understanding the local as well as global features efficiently.

The previous literatures [4], [5], [7] related to wheat plant phenotyping presented laborious, destructive, and complex image processing pipelines for detecting and characterizing the spikes. Most of the image processing pipelines involve the color intensity thresholding technique. [6], [7] presented non-destructive and feature based segmentation to characterize the spikes but, the features were manually defined. Recently, some researchers [3], [6], [8] proposed computer vision based approaches by combining the digital image processing and deep-learning technique for auto-detecting spikes non-destructively. But, there is a very limited easy-to-use pipeline available for detecting and characterizing spikes from the visual image of wheat plant. In this context, our main focus is to develop an online, easy-to-use, generalized and robust platform to characterize wheat spikes non-destructively.

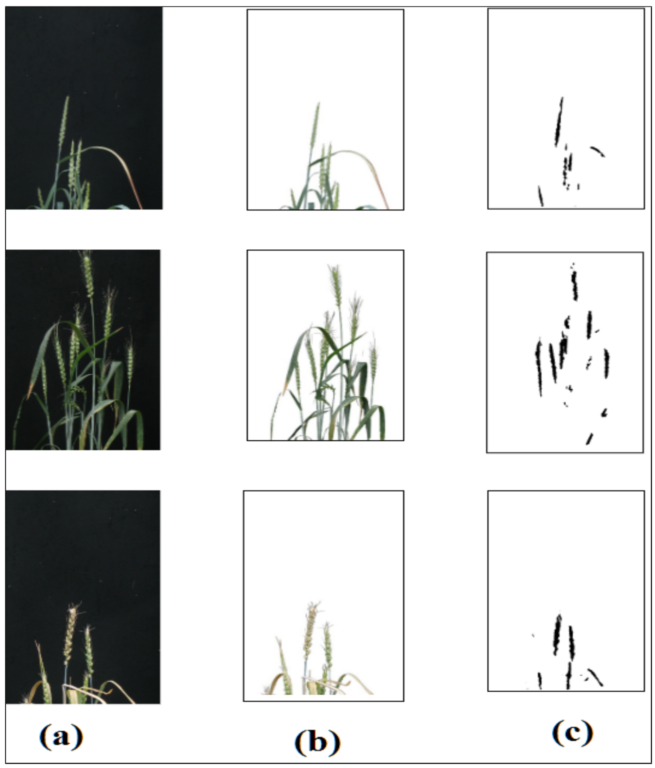

FIGURE 6. Comparative study with ACID (Annotated Crop Image Dataset) dataset available at https://plantimages.nottingham.ac.uk/: (a) test images (b) black background converted into white (c) detected spikes using Web-SpikeSegNet software

TABLE 5. Segmentation performance analysis of the ACID dataset

Type I Error	Type II Error	Accuracy	Precision	Recall	F1 Score
0.00164	0.0576	0.9955	0.9962	0.9958	0.9962

IV. CONCLUSIONS

Recognition and counting of spikes for the large set of germplasms in a non-destructive way is an enormously challenging task. This study developed web-based software "Web-SpikeSegNet" using the robust SpikeSegNet approach, which is based on digital image analysis and deep-learning techniques. The software is freely available for researchers, and students are working particularly in the field of wheat plant phenotyping. Further, it is a useful tool in the automated phenomics facility to automate the phenology-based treatment. Web-SpikeSegNet is a significant step toward studying the wheat crop yield phenotyping and can be extended to the other cereal crops.

REFERENCES

[1] J. R. Porter and S. Christensen, "Deconstructing crop processes and models via identities," Plant, cell & environment, vol. 36, no. 11, pp. 1919–1925, 2013.

[2] S. A. Tsaftaris, M. Minervini, and H. Scharr, "Machine learning for plant phenotyping needs image processing," Trends in plant science, vol. 21, no. 12, pp. 989–991, 2016.

[3] T. Misra, A. Arora, S. Marwaha, V. Chinnusamy, A. R. Rao, R. Jain, R. N. Sahoo, M. Ray, S. Kumar, D. Raju et al., "Spikesegnet-a deep learning approach utilizing encoder-decoder network with hourglass for spike segmentation and counting in wheat plant from visual imaging," Plant methods, vol. 16, no. 1, pp. 1–20, 2020.

[4] K. Bi, P. Jiang, L. Li, B. Shi, and C. Wang, "Non-destructive measurement of wheat spike characteristics based on morphological image processing," Transactions of the Chinese Society of Agricultural Engineering, vol. 26, no. 12, pp. 212–216, 2010.

[5] L. Qiongyan, J. Cai, B. Berger, M. Okamoto, and S. J. Miklavcic, "Detecting spikes of wheat plants using neural networks with laws texture energy," Plant Methods, vol. 13, no. 1, p. 83, 2017.

[6] M. P. Pound, J. A. Atkinson, D. M. Wells, T. P. Pridmore, and A. P. French, "Deep learning for multi-task plant phenotyping," in Proceedings of the IEEE International Conference on Computer Vision Workshops, 2017, pp. 2055–2063.

[7] P. Sadeghi-Tehran, K. Sabermanesh, N. Virlet, and M. J. Hawkesford, "Automated method to determine two critical growth stages of wheat: heading and flowering," Frontiers in Plant Science, vol. 8, p. 252, 2017.

[8] M. M. Hasan, J. P. Chopin, H. Laga, and S. J. Miklavcic, "Detection and analysis of wheat spikes using convolutional neural networks," Plant Methods, vol. 14, no. 1, p. 100, 2018.

[9] E. David, S. Madec, P. Sadeghi-Tehran, H. Aasen, B. Zheng, S. Liu, N. Kirchgessner, G. Ishikawa, K. Nagasawa, M. A. Badhon et al., "Global wheat head detection (gwhd) dataset: a large and diverse dataset of high-resolution rgb-labelled images to develop and benchmark wheat head detection methods," Plant Phenomics, vol. 2020, 2020.

[10] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, Deep learning. MIT press Cambridge, 2016, vol. 1.

[11] H. Xiong, Z. Cao, H. Lu, S. Madec, L. Liu, and C. Shen, "Tasselnetv2: in-field counting of wheat spikes with context-augmented local regression networks," Plant Methods, vol. 15, no. 1, pp. 1–14, 2019.

[12] P. Sadeghi-Tehran, N. Virlet, E. M. Ampe, P. Reyns, and M. J. Hawkesford, "Deepcount: in-field automatic quantification of wheat spikes using simple linear iterative clustering and deep convolutional neural networks," Frontiers in plant science, vol. 10, p. 1176, 2019.

[13] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in International Conference on Medical image computing and computer-assisted intervention. Springer, 2015, pp. 234–241.

[14] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," IEEE transactions on pattern analysis and machine intelligence, vol. 39, no. 12, pp. 2481–2495, 2017.

[15] R. R. Jha, G. Jaswal, D. Gupta, S. Saini, and A. Nigam, "Pixisegnet: Pixel-level iris segmentation network using convolutional encoder–decoder with stacked hourglass bottleneck," IET Biometrics, vol. 9, no. 1, pp. 11–24, 2019.

[16] T. Berners-Lee, "Tim berners-lee," Bloomberg Businessweek, 1989.

[17] E. A. Meyer, Cascading style sheets: The definitive guide. " O'Reilly Media, Inc.", 2004.

[18] G. Mainland, M. Welsh, and G. Morrisett, "Flask: A language for data-driven sensor network programs," Harvard Univ., Cambridge, MA, Tech. Rep. TR-13-06, 2006.

[19] S. Yehuda and S. Tomer, "Advanced javascript programming," BPB Publication, New Delhi India, 1998.

[20] T. Hope, Y. S. Resheff, and I. Lieder, Learning tensorflow: A guide to building deep learning systems. " O'Reilly Media, Inc.", 2017.

[21] A. Gulli and S. Pal, Deep learning with Keras. Packt Publishing Ltd, 2017.

[22] E. Bressert, SciPy and NumPy: an overview for developers. " O'Reilly Media, Inc.", 2012.

[23] E. A. Christensen, F. J. Blanco-Silva et al., Learning SciPy for numerical and scientific computing. Packt Publishing Ltd, 2015.

[24] S. Tosi, Matplotlib for Python developers. Packt Publishing Ltd, 2009.

[25] J. Howse, OpenCV computer vision with python. Packt Publishing Ltd, 2013.

[26] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A convolutional neural network for modelling sentences," arXiv preprint arXiv:1404.2188, 2014.

[27] F. Agostinelli, M. Hoffman, P. Sadowski, and P. Baldi, "Learning activation functions to improve deep neural networks," arXiv preprint arXiv:1412.6830, 2014.

[28] B. Graham, "Fractional max-pooling," arXiv preprint arXiv:1412.6071, 2014.

[29] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," arXiv preprint arXiv:1409.1556, 2014.

[30] S. P. Mohanty, D. P. Hughes, and M. Salathé, "Using deep learning for image-based plant disease detection," Frontiers in plant science, vol. 7, p. 1419, 2016.

[31] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," arXiv preprint arXiv:1603.07285, 2016.

[32] W. Liu, Y. Wen, Z. Yu, and M. Yang, "Large-margin softmax loss for convolutional neural networks." in ICML, vol. 2, no. 3, 2016, p. 7.

[33] M. D. Abràmoff, P. J. Magalhães, and S. J. Ram, "Image processing with imagej," Biophotonics international, vol. 11, no. 7, pp. 36–42, 2004.

[34] A. Asundi and Z. Wensen, "Fast phase-unwrapping algorithm based on a gray-scale mask and flood fill," Applied optics, vol. 37, no. 23, pp. 5416–5420, 1998.

[35] H. Proença, S. Filipe, R. Santos, J. Oliveira, and L. A. Alexandre, "The ubiris. v2: A database of visible wavelength iris images captured on-the-move and at-a-distance," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 32, no. 8, pp. 1529–1535, 2009.

[36] M. Haindl and M. Krupička, "Unsupervised detection of non-iris occlusions," Pattern Recognition Letters, vol. 57, pp. 60–65, 2015.

[37] Z. Zhao and K. Ajay, "An accurate iris segmentation framework under relaxed imaging constraints using total variation model," in Proceedings of the IEEE international conference on computer vision, 2015, pp. 3828–3836.