Web-SpikeSegNet: Deep Learning Framework for Recognition and Counting of Spikes from Visual Images of Wheat Plants

Tanuj Misra
RLBCAU: Rani Lakshmi Bai Central Agricultural University
Alka Arora (alka.arora@icar.gov.in)
Indian Agricultural Statistics Research Institute https://orcid.org/0000-0003-0999-1077
Sudeep Marwaha
Indian Agricultural Statistics Research Institute
Ranjeet Ranjan Jha
Indian Institute of Technology Mandi
Mrinmoy Ray
Indian Agricultural Statistics Research Institute
Eldho Varghese
CMFRI: Central Marine Fisheries Research Institute
Sudhir Kumar
Indian Agricultural Research Institute
Aditya Nigam
Indian Institute of Technology Mandi
Rabi Narayan Sahoo
Indian Agricultural Research Institute
Viswanathan Chinnusamy
Indian Agricultural Research Institute

Software

Keywords: computer vision, deep learning, image analysis, spike detection and counting, Web-SpikeSegNet, wheat

Posted Date: February 17th, 2021

DOI: https://doi.org/10.21203/rs.3.rs-230449/v1

SOFTWARE

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

Tanuj Misra[1,2]
, Alka Arora[1*]
, Sudeep Marwaha[1]
, Ranjeet Ranjan Jha[3]
, Mrinmoy Ray[1]
, Eldho Varghese[4]
, Sudhir Kumar[5]
, Aditya Nigam[3]
, Rabi Narayan Sahoo[5]
and Viswanathan Chinnusamy [5]

Correspondence:
Alka.Arora@icar.gov.in
Division of Computer Application, ICAR-Indian Agricultural Statistics Research Institute, New Delhi, India
Full list of author information is available at the end of the article

Abstract

Background: 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 of spike helps in identifying heading, and counting of the spikes as well as area of the spikes will be useful for determination of the yield of wheat plant. Hence detection and counting of spikes which is considered as the grain bearing organ, has great importance in the phenomics study of large sets of germplasms.

Results: 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. This platform implements the "SpikeSegNet" approach developed by Misra et al.(2020), which has proved as an effective and robust approach for spike detection and counting. Architecture of the Web-SpikeSegNet consists of 2 layers. First Layer, Client Side Interface Layer, deals with deals with end user's requests and its corresponding responses management while the second layer, Server Side Application Layer consisting of spike detection and counting module. The backbone of the spike detection module comprises of deep encoder-decoder network with hourglass for spike segmentation. 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 wheat plant's visual images using LemnaTec imaging platform installed at Nanaji Deshmukh Plant Phenomics Centre, ICAR-Indian Agricultural Research Institute, New Delhi, India 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%.

Conclusions: In this study, freely available web-based software has been developed based on combined digital image analysis and deep learning techniques. 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.

Keywords: computer vision; deep learning; image analysis; spike detection and counting; Web-SpikeSegNet; wheat

Background

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 through traditional methods using naked-eye is a tedious and time-consuming job. Presently, non-destructive image analysis based phenotyping is gaining momentum and proved as 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, 4, 5, 6, 7, 8]. In computer vision, the problem of spike detection lies under the domain of pixel-wise segmentation of object. [4], [5] and [7] used manually defined color intensities and textures for spike segmentation. Pound et al. (2017)[6] and Hasan et al. (2018) [8] used Autoencoder [9] and Region-based Convolutional Neural Network (R-CNN) [9] deep-learning technique, respectively, to detect and characterize spikes with greater than 90 percent accuracy. 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 visual images of wheat plants for easy accessibility and quick reference. The developed web-solution has a wide application in the plant phenomics domain and will be useful for the 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/`.

Implementation

Web_SpikeSegNet is developed based on the approach give by Misra et al. (2020) [3]. The approch is based on 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 [10], SegNet [11], and PixISegNet [12], 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 for better segmentation of the spikes as given in Fig. 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.

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 the 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. Hyper Text Markup Language [13], Cascading Style Sheets [14], and JavaScript [15] technologies were used as a building block of CSIL which deals with the end-user's requests and its corresponding responses management. SSAL consists of two modules: spike detection and spike counting module. Spike detection module was developed using python libraries such as Tensorflow [16], Keras [17], Numpy [18], Scipy [19], Matplotlib [20] and OpenCV [21] for constructing and implementing the deep learning model.Convolutional encoder network [9] (Encoder_SpikeSegNet), decoder network [9] (Decoder_SpikeSegNet), and bottleneck network ([9], [12]) using stacked hourglasses (Bottleneck_SpikeSegNet) are the backbone of LPNet, GMRNet and correspondingly 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 [22] with a varying number of filters (16, 64, 128) followed by ReLU [23] and max-pooling layer with a window size of 2*2 [24]. Square filters are popularly used in state-of-art methods [25], and the mentioned window size is considered as standard [10, 26].Batch Normalization, a statistical procedure, is done to improve the performance as well as stability of the network. Decoder_SpikeSegNet network facilitates a special operation called transpose convolution [27], 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" (liu2016large) activation layer for classifying objects (or spikes).Bottleneck_SpikeSegNet network contains three hourglasses, which provide more confident segmentation by concentrating the most essential features captured at various occlusions, scale, and viewpoints [10, 8]. 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, number of filters) 128, 128, and 256 respectively, estimated empirically on the basis of optimal performances. Algorithms for implementing Encoder_SpikeSegNet, Decoder _SpikeSegNet, Bottleneck _SpikeSegNet, LPNet, and GMRNet are presented in Algorithm 1, 2, 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 [28] 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 [29] for counting of object.

Algorithm 1 Encode_SpikeSegNet: Encoding operation of SpikeSegNet

I: Input image/feature
Conv(input feature, $filter_$size, no. of filters): Convolution operation ▷ for generating feature maps
BatchNorm(): Batch normalization operation ▷ for improving the performance as well as stability of the network
Pool(): Pooling operation or down-sampling with window size 2*2
procedure ENCODER_SPIKESEGNET(I) ▷ input image of size 256*256
    //First Encoder Block
    $E\_conv\_1\_1 \leftarrow Conv$(I, $3*3, 16$) ▷ generates 16 feature maps of size 256*256
    $E\_batch\_1\_1 \leftarrow BatchNorm$(E_conv_1 _1) ▷ batch normalization of the features
    $E\_conv\_1\_2 \leftarrow Conv$(E_batch_1 _1, $3*3, 16$) ▷ generates 16 feature maps from the batch normalized features
    $E\_batch\_1\_2 \leftarrow BatchNorm$(E_conv_1_2)
    $I\_Encoded\_block\_1 \leftarrow Pool$(E_batch_1_2) ▷ size of each feature map reduced by half and returns 16 feature maps of size 128*128
    //Second Encoder Block. Here input is the output of First encoder block
    $E\_conv\_2\_1 \leftarrow Conv$(I_Encoded_block _1, $3*3, 64$) ▷ generates 64 feature maps of size 128*128
    $E\_batch\_2\_1 \leftarrow BatchNorm$(E_conv_2 _1) ▷ batch normalization of the features
    $E\_conv\_2\_2 \leftarrow Conv$(E_batch_2 _1, $3*3, 64$)
    $E\_batch\_2\_2 \leftarrow BatchNorm$(E_conv_2 _2)
    $I\_Encoded\_block\_2 \leftarrow Pool$(E_batch_2 _2) ▷ return 64 feature maps of size 64*64
    //Third Encoder Block. Here input is the output of second encoder block
    $E\_conv\_3\_1 \leftarrow Conv$(I_Encoded_block _2, $3*3, 128$) ▷ generates 128 feature maps of size 64*64
    $E\_batch\_3\_1 \leftarrow BatchNorm$(E_conv_3 _1)
    $E\_conv\_3\_2 \leftarrow Conv$(E_batch_3 _1, $3*3, 128$)
    $E\_batch\_3\_2 \leftarrow BatchNorm$(E_conv_3 _2)
    $I\_Encoded\_block\_3 \leftarrow Pool$(E_batch_3 _2) ▷ return 128 feature maps of size 32*32
return *I_Encoded_block _3*

Performance measurement of Web-SpikeSegNet

For evaluating the segmentation performance to detect the spikes, 100 wheat plant's visual images were captured and tested. Images were acquired using the LemnaTec facility installed at Nanaji Deshmukh Plant Phenomics Center, New Delhi, India. For the purpose, 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 [30, 31, 32]:

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}) image of size p×q,

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

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

$$E_1 = \frac{1}{n}\sum_{r=1}^{n} Pix\_Err_{\text{r}} \tag{2}$$

Algorithm 2 Decoder_SpikeSegNet: Decoding operation of SpikeSegNet

I: Output of Bottleneck_SpikeSegNet (for LPNet) or, output of Encoder_SpikeSegNet (for GMRNet).
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 ▷ to up-sample the feature maps
Merge(): Merge/concatenation operation ▷ for transferring the spatial information across the network
procedure DECODER_SPIKESEGNET(I) ▷ here input is 128 feature maps of size 32*32
    //First Decoder Block
    $T\_conv\_1 \leftarrow Tr\_Conv$(I, $3*3, 128$) ▷ Up-sampling done and return 128 decoded feature maps of size 64*64
    $D\_batch\_1\_1 \leftarrow BatchNorm$(T_conv_1) ▷ batch normalization of the features
    $M\_1 \leftarrow Merge$(D_batch_1 _1, I_Encoded_block _3) ▷ concatenation operation with the output of third Encoder block [refer Algorithm 1 Line no.: 23]
    $D\_conv\_1\_1 \leftarrow Conv$(M_1, $3*3, 128$)
    $D\_batch\_1\_2 \leftarrow BatchNorm$(D_conv_1 _1) ▷ batch normalization of the features
    $D\_conv\_1\_2 \leftarrow Conv$(D_batch_1 _1, $3*3, 128$)
    $I\_Decoded\_block\_1 \leftarrow BatchNorm$(D_conv_1 _2) ▷ Output of the 1st Decoder Block is 128 decoded feature maps of size 64*64
    //Second Decoder Block. Here input is the output of First Decoder block
    $T\_conv\_2 \leftarrow Tr\_Conv$(I_Decoded_block _1, $3*3, 64$) ▷ Up-sampling done and return 64 decoded feature maps of size 128*128
    $D\_batch\_2\_1 \leftarrow BatchNorm$(T_conv_2) ▷ batch normalization of the features
    $M\_2 \leftarrow Merge$(D_batch_2 _1, I_Encoded_block _2) ▷ concatenation operation with the output of second Encoder block [refer Algorithm 1 Line no.: 17]
    $D\_conv\_2\_1 \leftarrow Conv$(M_2, $3*3, 64$)
    $D\_batch\_2\_2 \leftarrow BatchNorm$(D_conv_2 _1)
    $D\_conv\_2\_2 \leftarrow Conv$(D_batch_2 _2, $3*3, 64$)
    $I\_Decoded\_block\_2 \leftarrow BatchNorm$(D_conv_2 _2) ▷ Output of the second Decoder Block is 64 decoded feature maps of size 128*128
    //Third Decoder Block. Here input is the output of Second Decoder block
    $T\_conv\_3 \leftarrow Tr\_Conv$(I_Decoded_block _2, $3*3, 16$) ▷ Up-sampling done and return 16 decoded feature maps of size 256*256
    $D\_batch\_3\_1 \leftarrow BatchNorm$(T_conv_3) ▷ batch normalization of the features
    $M\_3 \leftarrow Merge$(D_batch_3 _1, I_Encoded_block _1) ▷ concatenation operation with the output of First Encoder block [refer Algorithm 1 Line no.: 11]
    $D\_conv\_3\_1 \leftarrow Conv$(M_3, $3*3, 16$)
    $D\_batch\_3\_2 \leftarrow BatchNorm$(D_conv_3 _1)
    $D\_conv\_3\_2 \leftarrow Conv$(D_batch_3 _2, $3*3, 16$)
    $I\_Decoded\_block\_3 \leftarrow BatchNorm$(D_conv_3 _2) ▷ Output of the third Decoder Block is 16 decoded feature maps of size 256*256
return *I_Decoded_block _3*

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 ${E_2}^{\text{r}}$ is computed by the average of false-positives (FPR) and false negatives (FNR) rates at the pixel level defined as:

$$E_2^r = 0.5 * FPR + 0.5 * FNR \tag{3}$$

Where,

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

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
    $H\_1 \leftarrow$ HOURGLASS_SPIKESEGNET(I) ▷ Call HOURGLASS_SPIKESEGNET procedure and return, 128 feature maps of size 32*32
    $Scale\_up\_ \leftarrow$ SCALE_UP(H_1)▷ Call SCALE_UP procedure and return, 128 feature maps of size 64*64
    $H\_2 \leftarrow$ HOURGLASS_SPIKESEGNET($Scale\_up$)
    $Scale\_down\_ \leftarrow$ SCALE_DOWN(H_2) ▷ Call SCALE_DOWN procedure and return, 128 feature maps of size 32*32
    $H\_3\_ \leftarrow$ 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)
    $res\_1 \leftarrow$ RESIDUAL_BL(I) ▷ returns, 256 feature maps of size 32*32
    $pool_1 \leftarrow$Pool(res_1) ▷ down-sampling done and returns, 256 feature maps of size 16*16
    $res\_2 \leftarrow$ RESIDUAL_BL($pool\_1$) ▷ returns, 256 feature maps of size 16*16
    $pool_2 \leftarrow$Pool(res_2) ▷ down-sampling done and returns, 256 feature maps of size 8*8
    $res\_3 \leftarrow$ RESIDUAL_BL($pool\_2$) ▷ returns, 256 feature maps of size 8*8
    $pool_3 \leftarrow$Pool(res_3) ▷ down-sampling done and returns, 256 feature maps of size 4*4
    $res\_4 \leftarrow$ RESIDUAL_BL($pool\_3$) ▷ returns, 256 feature maps of size 4*4
    $res\_5 \leftarrow$ RESIDUAL_BL(res_4)
    $T_conv_1 \leftarrow$Tr_conv(res_5, $3*3$, 256) ▷ up-sampling done and returns, 256 feature maps of size 8*8
    $M_1 \leftarrow$Merge(T_conv_1, res_3)
    $res\_6 \leftarrow$ RESIDUAL_BL(M_1) ▷ returns, 256 feature maps of size 8*8
    $T_conv_2 \leftarrow$Tr_conv(res_6, $3*3$, 256) ▷ up-sampling done and returns, 256 feature maps of size 16*16
    $M_2 \leftarrow$Merge(T_conv_2, res_2)
    $res\_7 \leftarrow$ RESIDUAL_BL(M_2) ▷ returns, 256 feature maps of size 16*16
    $T_conv_3 \leftarrow$Tr_conv(res_7, $3*3$, 256) ▷ up-sampling done and returns, 256 feature maps of size 32*32
    $M_3 \leftarrow$Merge(T_conv_3, res_1)
    $res\_8 \leftarrow$ RESIDUAL_BL(M_3) ▷ returns, 256 feature maps of size 32*32
return res_8
procedure RESIDUAL_BL(I)
    $res_conv_1 \leftarrow$Conv(I, $1*1$, 128)
    $res_conv_2 \leftarrow$Conv(res_conv_1, $3*3$, 128)
    $res_conv_3 \leftarrow$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)
    $sc_up_conv_1 \leftarrow$Conv(I, $3*3, 128$)
    $sc_up_batch_1 \leftarrow$BatchNorm($sc\_up\_conv\_1$)
    $sc_up_conv_2 \leftarrow$Conv($sc\_up\_batch\_1$, $3*3, 128$)
    $sc_up_batch_2 \leftarrow$BatchNorm($sc\_up\_conv\_2$)
    $sc_up_pool \leftarrow$Tr _Pool($sc\_up\_batch\_2$)
return sc_up_pool
procedure SCALE_DOWN(I)
    $sc_down_pool_1 \leftarrow$Pool($I$)
    $sc_down_conv_1 \leftarrow$BatchNorm($sc\_down\_pool\_1$, $3*3$, 128)
    $sc_down_batch_1 \leftarrow$BatchNorm($sc\_down\_conv\_1$)
    $sc_down_conv_2 \leftarrow$Conv($sc\_down\_batch\_1$, $3*3$, 128)
    $sc_down_batch_2 \leftarrow$BatchNorm($sc\_down\_conv\_2$)
return $sc\_down\_batch\_2$

Algorithm 4 LPNet Local Patch Extraction Network

1: **I**: Input image/feature
2: **procedure** LPNET(I) ▷ here input is visual image patches of size 256*256
3: $Encoded\_I \leftarrow$ ENCODER_SPIKESEGNET(I) ▷ Call Algorithm 1. Return encoded feature maps of the input image
4: $Bottleneck\_I \leftarrow$ BOTTLENECK_SPIKESEGNET($Encoded\_I$) ▷ Call Algorithm 3. Return refined feature maps of the input features
5: $Decoded\_I \leftarrow$ DECODER_SPIKESEGNET($Bottleneck\_I$) ▷ Call Algorithm 2. Return decoded feature maps of the input features
6: **return** $Decoded\_I$ ▷ Segmeted mask image of size 256*256 containing spikes regions corresponding to the input patches.

Algorithm 5 GMRNet

1: **I**: Input image/feature
2: **procedure** GMRNET(I) ▷ here input is the output image/feature of LPNet
3: $Encoded\_I \leftarrow$ ENCODER_SPIKESEGNET(I) ▷ Call Algorithm 1. Return encoded feature maps of the input image
4: $Decoded\_I \leftarrow$ DECODER_SPIKESEGNET($Encoded\_I$) ▷ Call Algorithm 2. Return decoded feature maps of the input features
5: **return** $Decoded\_I$ ▷ Refined segmeted mask image of size 256*256 containing spikes regions corresponding to the input image/feature.

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

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

$$E_2 = \frac{1}{n} \sum_{r=1}^{n} E_2^r \tag{6}$$

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:

$$Precision = TP/(TP + FP) \tag{7}$$

measures the percentage of detected pixels are actually spikes

$$Recall = TP/(TP + FN) \tag{8}$$

measures the percentage of actually spikes spike pixels are detected

$$Accuracy = (TP + TN)/(TP + TN + FP + FN) \tag{9}$$

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

Results

To demonstrate the working environment of Web-SpikeSegNet, a case study is presented here. For this purpose, sample images of the wheat plant grown in the pot were collected using the LemnaTec imaging sensor installed at Nanaji Deshmukh Plant Phenomics Centre, ICAR-IARI, New Delhi, India. The architecture of Web-SpikeSegNet mentioned in section 3, and 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 plant 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:

1 Select and upload visual image of wheat plant of size 1656*1356 consisting of above ground parts only (Fig. 4) as discussed in [3].
2 Click on "Generate Patches" button for dividing the whole image into patches (Fig. 5). 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 as shown in (Fig. 6).
3 Click on "Run LPNet" to run the LPNet module for extracting contextual and spatial features at patch level (Fig. 7). 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(Fig. 8).
5 For counting the wheat spikes, click on "Count"button and the corresponding spikes count will be displayed on the next window (Fig. 9).

Performance analysis of Web-SpikeSegNet

Segmentation performances of the Web-SpikeSegNet for the mentioned statistical parameters (eq. 1 to eq. 10) are computed and the average values are presented in Table 1.As the performance of spike detection is calculated at pixel level, the value of E1 (=0.00159) depict that on an average only 104 pixels are mis-classified among 65,536 pixels which is the pixel size of one image i.e., 65,536 (256 * 256). 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%.

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 the 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.

Acknowledgements

The first author acknowledges the fellowship received from ICAR-IASRI, New Delhi, India to undertake this research work as part of Ph.D. Also acknowledges Nanaji Deshmukh Plant Phenomics Facility, ICAR-IARI, New Delhi-12 . . .

Funding

This work was supported by National Agriculture Science Fund (NASF), ICAR, Grant No. NASF/Phen-6005/2016-17 and NAHEP world bank funded project.. . .

Abbreviations

CSIL: Client Side Interface Layer.
SSAL: Server Side Application Layer.
LPNet:Local Patch Extraction Network
GMRNet:Global Mask Refinement Network
Encoder_SpikeSegNet:Encoder Network of SpikeSegNet
Decoder_SpikeSegNet:Decoder Network of SpikeSegNet
Bottleneck_SpikeSegNet:Bottleneck Network of SpikeSegNet

Availability of data and materials

The datasets used and/or analysed during the current study are available from the corresponding author on reasonable request.. . .

Ethics approval and consent to participate

Not applicable.. . .

Competing interests

The authors declare that they have no competing interests.

Consent for publication

Consent and approval for publication from all the authors was obtained.. . .

Authors' contributions

TM: Conceptualization, image collection, methodology proposed and developing, computer programming, supervision, validation, writing original draft, writing—review and editing of draft.
AA: Conceptualization, methodology proposed and developing, supervision, writing—review and editing of draft.
SM: Conceptualization, methodology proposed and developing, supervision.
RRJ: Methodology proposed and developing, computer programming, validation.
MR: Validation, computer programming.
EV: Writing—review and editing of draft.
SK: Image collection. DR: Image collection.
AN: Methodology proposed and developing.
RNS: Writing—review and editing of draft.
VC: conceptualization, writing—review and editing of draft. . . .

Authors' information

Text for this section. . .

Author details

[1]Division of Computer Application, ICAR-Indian Agricultural Statistics Research Institute, New Delhi, India. [2]Computer Science, Rani Lakshmi Bai Central Agricultural University, Jhansi, India. [3]School of Computing and Electrical Engineering (SCEE), Indian Institute of Technology Mandi, Mandi, India. [4], ICAR-Central Marine Fisheries Research Institute, Kochi-682 0185, Kochi, India. [5], ICAR-Indian Agricultural Research Institute, Library Avenue, New Delhi 110 012, New Delhi, India.

References

1. Porter, J.R., Christensen, S.: Deconstructing crop processes and models via identities. Plant, cell & environment **36**(11), 1919–1925 (2013)
2. Tsaftaris, S.A., Minervini, M., Scharr, H.: Machine learning for plant phenotyping needs image processing. Trends in plant science **21**(12), 989–991 (2016)
3. Misra, T., Arora, A., Marwaha, S., Chinnusamy, V., Rao, A.R., Jain, R., Sahoo, R.N., Ray, M., Kumar, S., Raju, D., *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 **16**(1), 1–20 (2020)
4. Bi, K., Jiang, P., Li, L., Shi, B., Wang, C.: Non-destructive measurement of wheat spike characteristics based on morphological image processing. Transactions of the Chinese Society of Agricultural Engineering **26**(12), 212–216 (2010)
5. Qiongyan, L., Cai, J., Berger, B., Okamoto, M., Miklavcic, S.J.: Detecting spikes of wheat plants using neural networks with laws texture energy. Plant Methods **13**(1), 83 (2017)
6. Pound, M.P., Atkinson, J.A., Wells, D.M., Pridmore, T.P., French, A.P.: Deep learning for multi-task plant phenotyping. In: Proceedings of the IEEE International Conference on Computer Vision Workshops, pp. 2055–2063 (2017)
7. Sadeghi-Tehran, P., Sabermanesh, K., Virlet, N., Hawkesford, M.J.: Automated method to determine two critical growth stages of wheat: heading and flowering. Frontiers in Plant Science **8**, 252 (2017)
8. Hasan, M.M., Chopin, J.P., Laga, H., Miklavcic, S.J.: Detection and analysis of wheat spikes using convolutional neural networks. Plant Methods **14**(1), 100 (2018)
9. Goodfellow, I., Bengio, Y., Courville, A., Bengio, Y.: Deep Learning vol. 1. MIT press Cambridge, ??? (2016)
10. Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation. In: International Conference on Medical Image Computing and Computer-assisted Intervention, pp. 234–241 (2015). Springer
11. Badrinarayanan, V., Kendall, A., Cipolla, R.: Segnet: A deep convolutional encoder-decoder architecture for image segmentation. IEEE transactions on pattern analysis and machine intelligence **39**(12), 2481–2495 (2017)
12. Jha, R.R., Jaswal, G., Gupta, D., Saini, S., Nigam, A.: Pixisegnet: Pixel-level iris segmentation network using convolutional encoder–decoder with stacked hourglass bottleneck. IET Biometrics **9**(1), 11–24 (2019)
13. Berners-Lee, T.: Tim berners-lee. Bloomberg Businessweek (1989)
14. Meyer, E.A.: Cascading Style Sheets: The Definitive Guide. " O'Reilly Media, Inc.", ??? (2004)
15. Yehuda, S., Tomer, S.: Advanced javascript programming. BPB Publication, New Delhi India (1998)
16. Hope, T., Resheff, Y.S., Lieder, I.: Learning Tensorflow: A Guide to Building Deep Learning Systems. " O'Reilly Media, Inc.", ??? (2017)
17. Gulli, A., Pal, S.: Deep Learning with Keras. Packt Publishing Ltd, ??? (2017)
18. Bressert, E.: SciPy and NumPy: an Overview for Developers. " O'Reilly Media, Inc.", ??? (2012)
19. Christensen, E.A., Blanco-Silva, F.J., *et al.*: Learning SciPy for Numerical and Scientific Computing. Packt Publishing Ltd, ??? (2015)
20. Tosi, S.: Matplotlib for Python Developers. Packt Publishing Ltd, ??? (2009)
21. Howse, J.: OpenCV Computer Vision with Python. Packt Publishing Ltd, ??? (2013)
22. Kalchbrenner, N., Grefenstette, E., Blunsom, P.: A convolutional neural network for modelling sentences. arXiv preprint arXiv:1404.2188 (2014)
23. Agostinelli, F., Hoffman, M., Sadowski, P., Baldi, P.: Learning activation functions to improve deep neural networks. arXiv preprint arXiv:1412.6830 (2014)
24. Graham, B.: Fractional max-pooling. arXiv preprint arXiv:1412.6071 (2014)
25. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
26. Mohanty, S.P., Hughes, D.P., Salathé, M.: Using deep learning for image-based plant disease detection. Frontiers in plant science **7**, 1419 (2016)
27. Dumoulin, V., Visin, F.: A guide to convolution arithmetic for deep learning. arXiv preprint arXiv:1603.07285 (2016)
28. Abràmoff, M.D., Magalhães, P.J., Ram, S.J.: Image processing with imagej. Biophotonics international **11**(7), 36–42 (2004)
29. Asundi, A., Wensen, Z.: Fast phase-unwrapping algorithm based on a gray-scale mask and flood fill. Applied optics **37**(23), 5416–5420 (1998)
30. Proença, H., Filipe, S., Santos, R., Oliveira, J., Alexandre, L.A.: 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 **32**(8), 1529–1535 (2009)
31. Haindl, M., Krupička, M.: Unsupervised detection of non-iris occlusions. Pattern Recognition Letters **57**, 60–65 (2015)
32. Zhao, Z., Ajay, K.: An accurate iris segmentation framework under relaxed imaging constraints using total variation model. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 3828–3836 (2015)

Figures

Tables

Table 1: 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 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

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.

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: Select and upload visual image of wheat plant

Figure 5: "Generate Patches" button is used for dividing the uploaded visual image into over-lapping patches

Figure 6: Overlapping patch generation from the uploaded visual image after clicking on "Generate Patches" button

Figure 7: Output of the LPNet are the segmented images corresponding to the patch images

Figure 8: The output of LPNet are merged and further refined at global level by clicking on "Run GMRNet" button

Figure 9: By clicking on "Count" button corresponding spikes count will be displayed

Figures

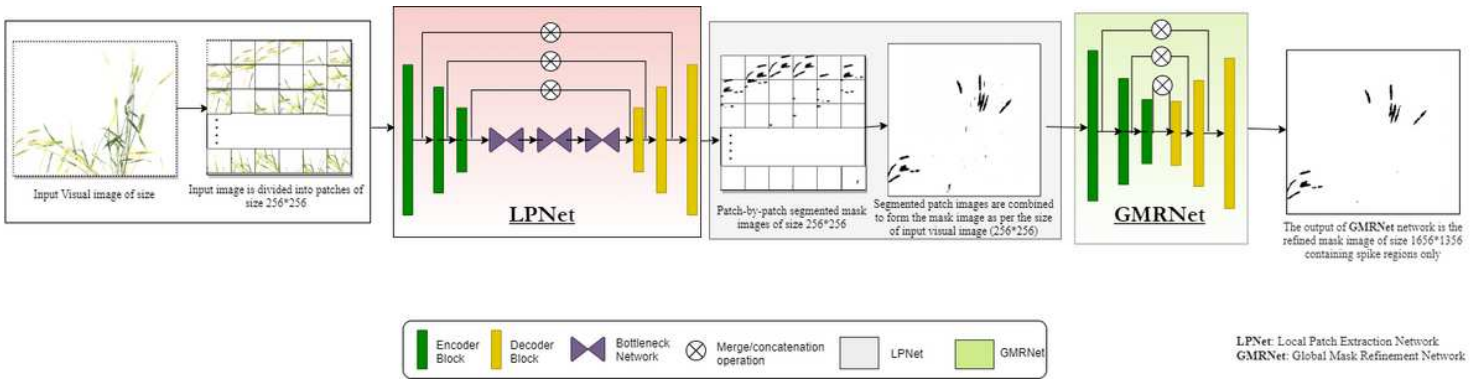

Figure 1

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

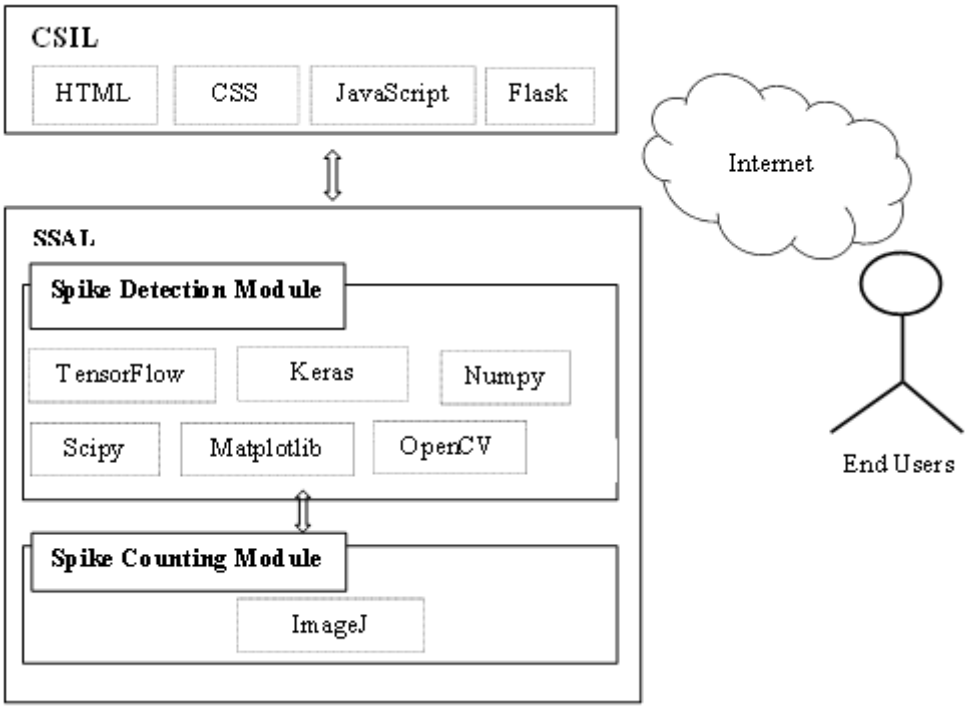

Figure 2

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.

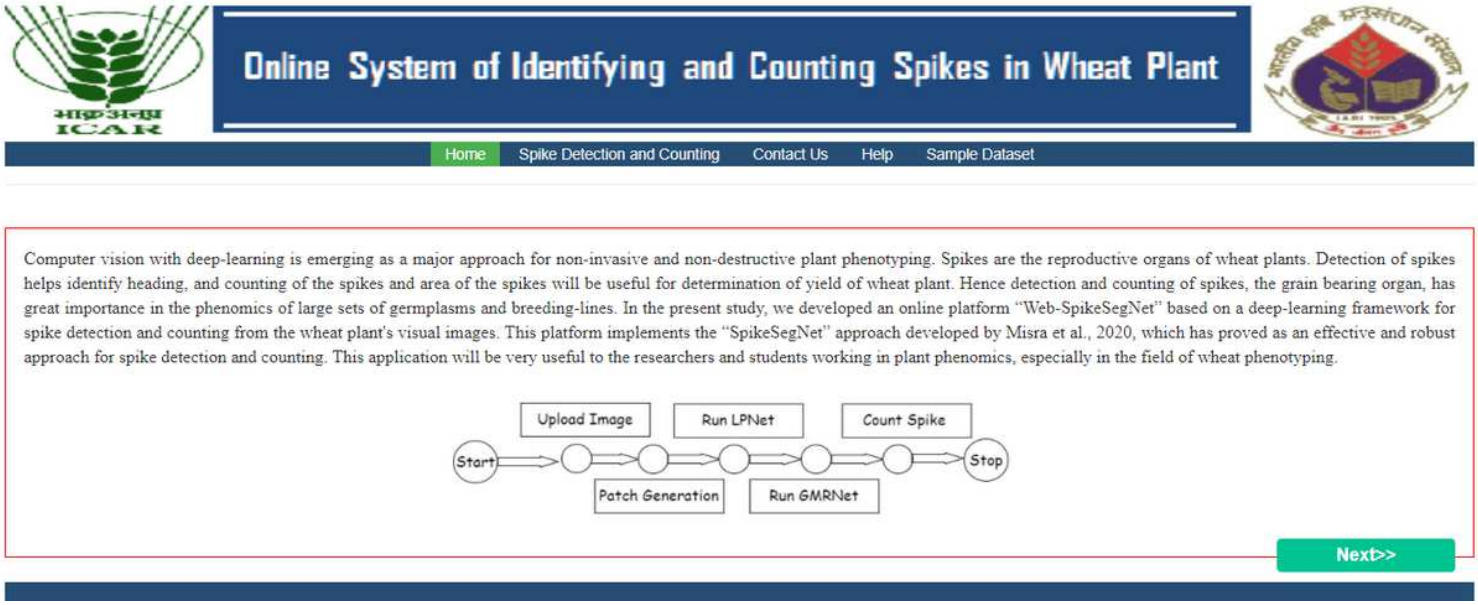

Figure 3

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

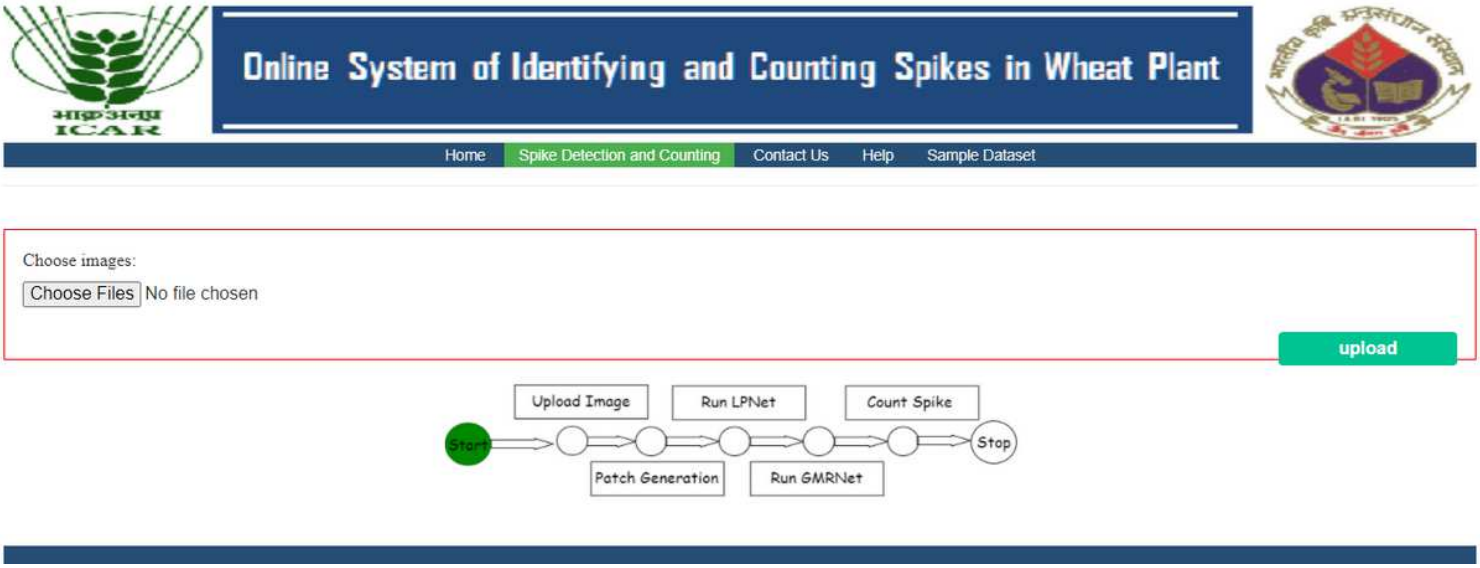

Figure 4

Select and upload visual image of wheat plant

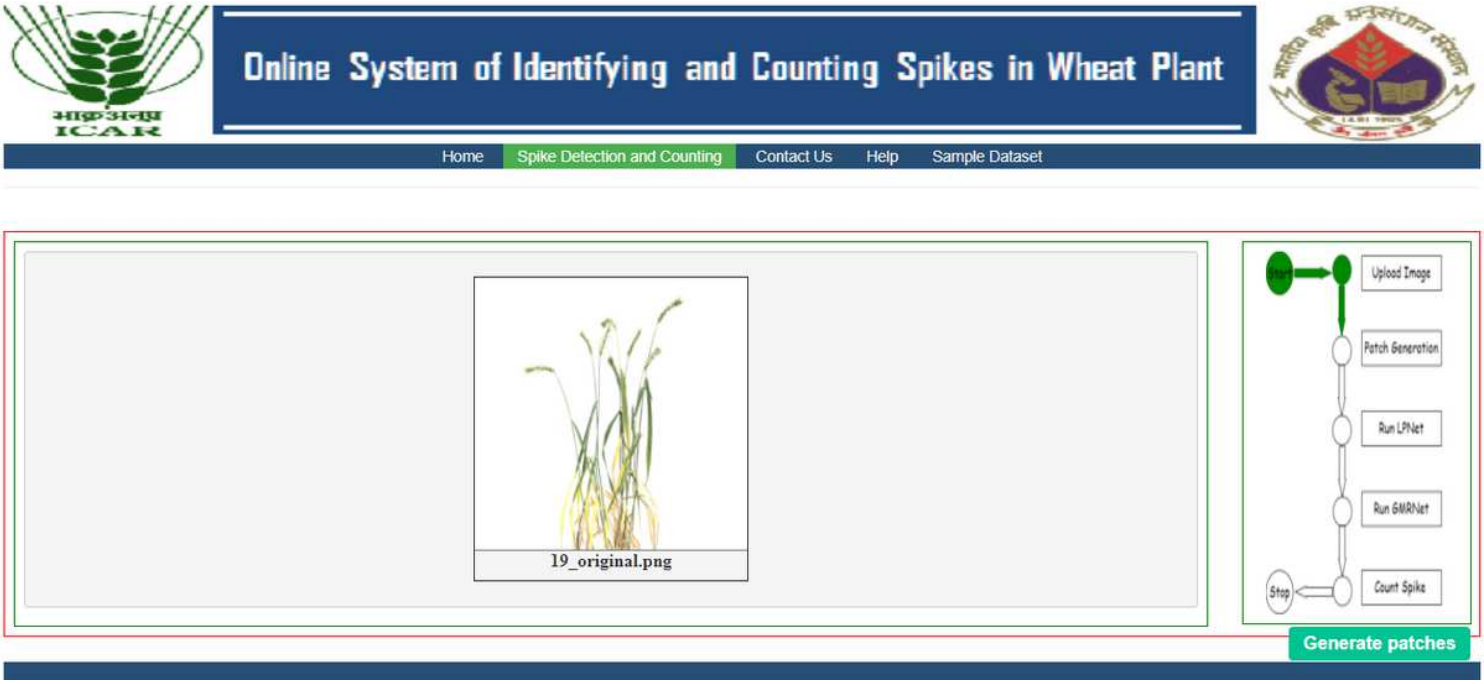

Figure 5

“Generate Patches” button is used for dividing the uploaded visual image into over-lapping patches

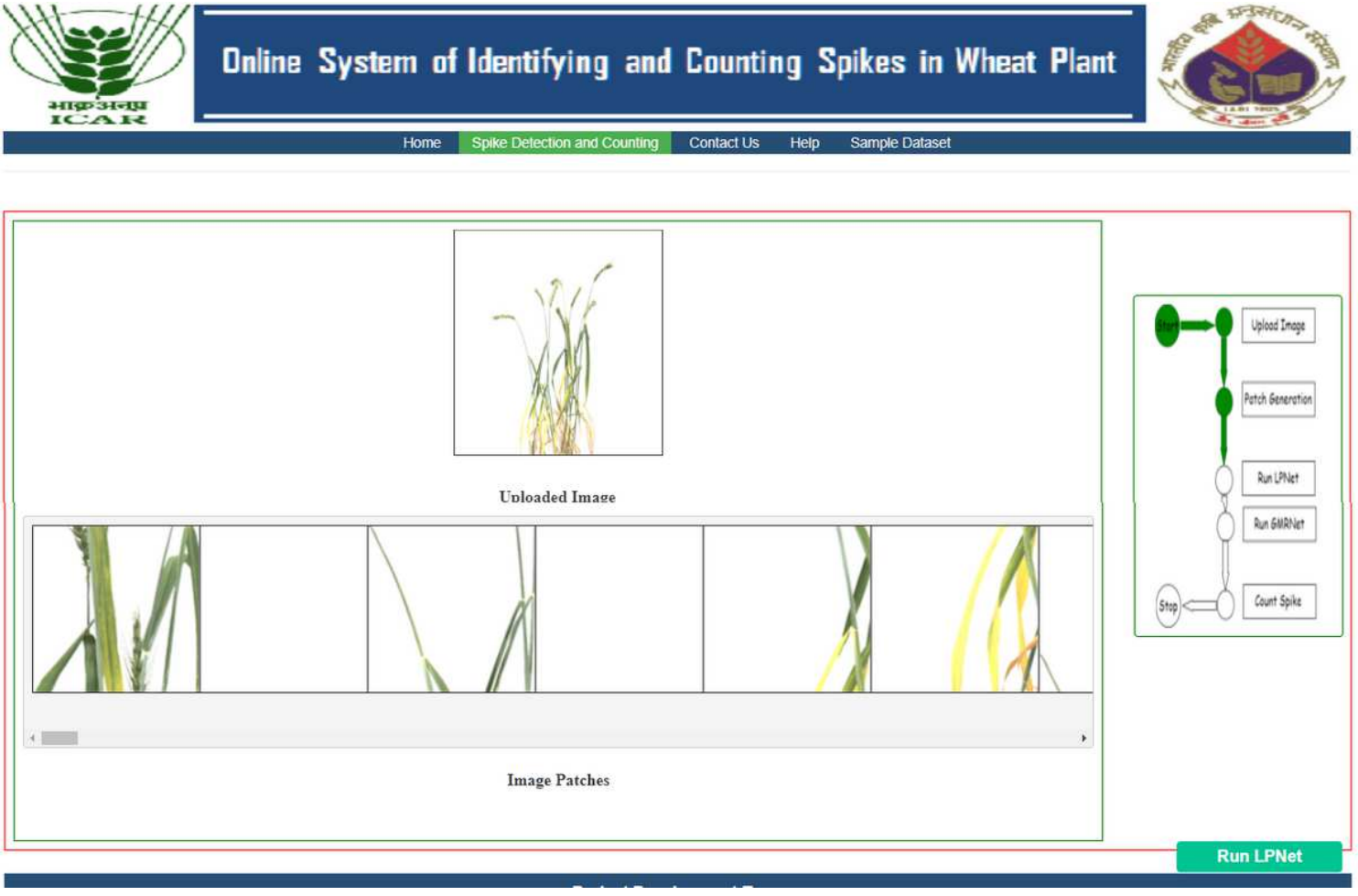

Figure 6

Overlapping patch generation from the uploaded visual image after clicking on “Generate Patches” button

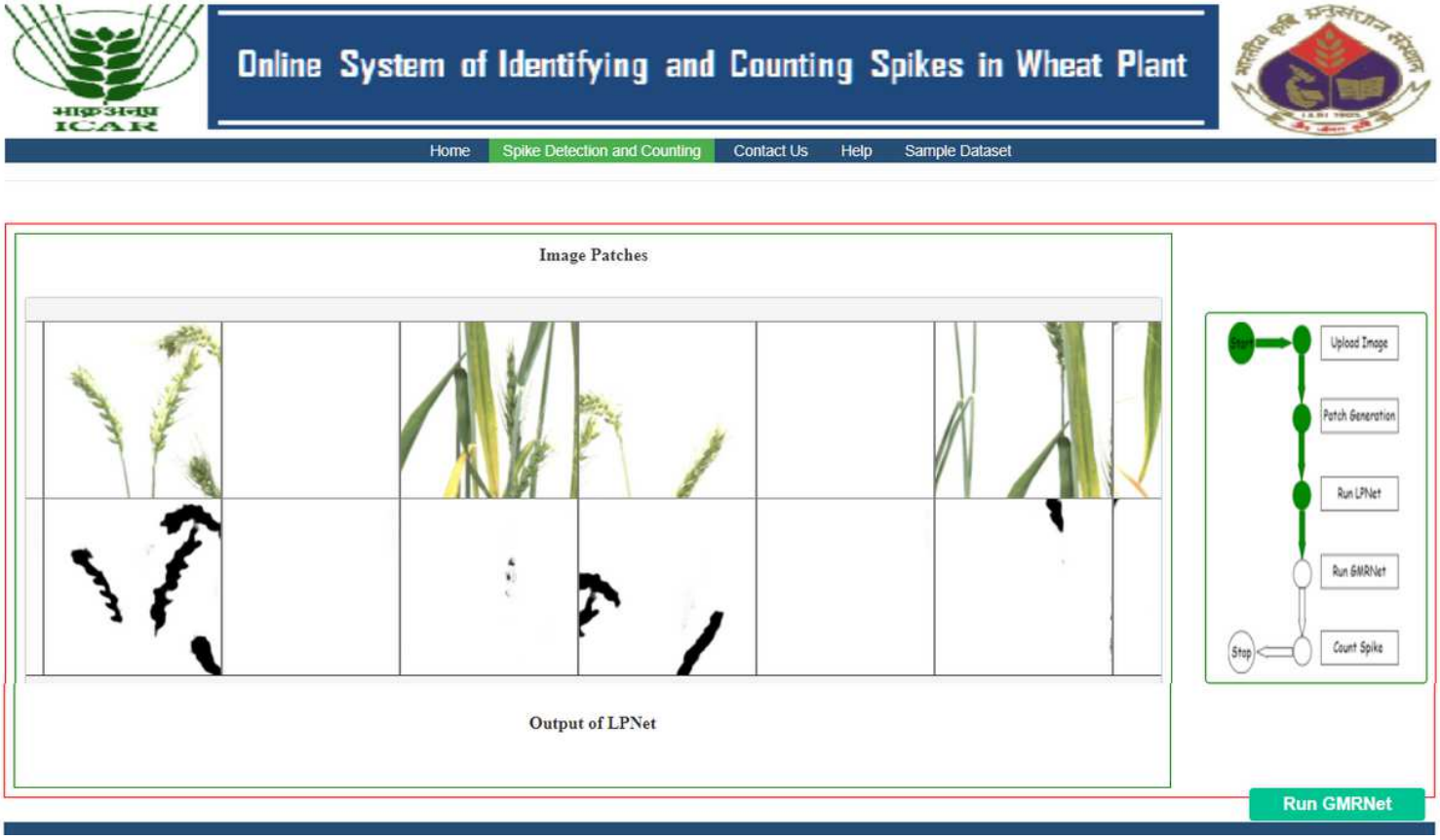

Figure 7

Output of the LPNet are the segmented images corresponding to the patch images

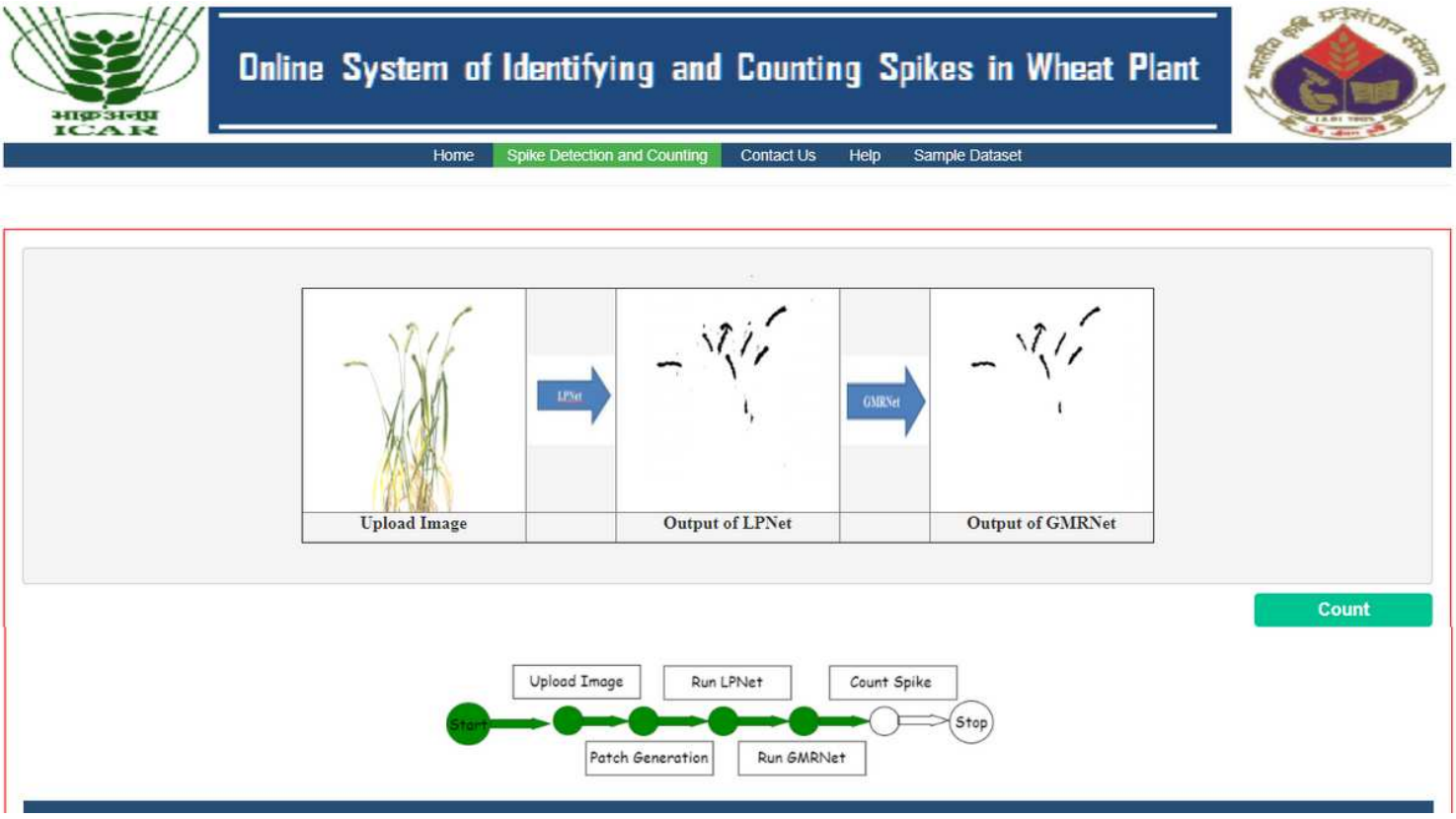

Figure 8

The output of LPNet are merged and further refined at global level by clicking on “Run GMRNet” button

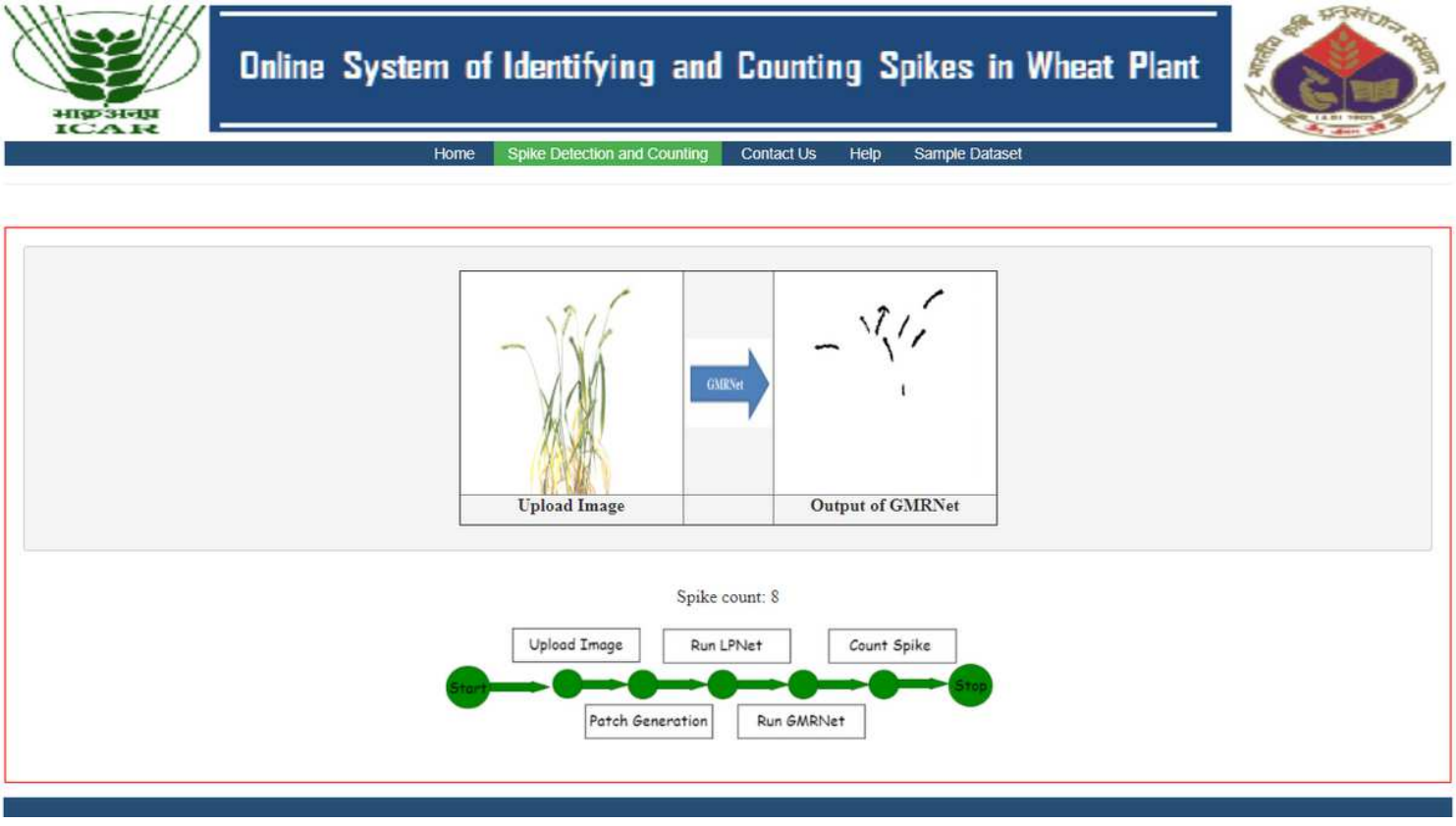

Figure 9

By clicking on “Count” button corresponding spikes count will be displayed