Nanotechnology Perceptions
ISSN 1660-6795
www.nano-ntp.com

Deepincepnet: Disease Detection in Corn or Maize Plant Leaves Using Specim Iq Hyperspectral Imaging and Proposed Dnn Classifiers with Inception Networks

Praba V[1], Dr. Krishnaveni K[2]

[1]*Research Scholar, Department of Computer Science, Sri S.Ramasamy Naidu Memorial College, (Affiliated to Madurai Kamaraj University, Madurai), Sattur,Tamilnadu, India, praba@srnmcollege.ac.in*
[2]*Associate Professor & Head, Department of Computer Science, Sri S. Ramasamy Naidu Memorial College, (Affiliated to Madurai Kamaraj University, Madurai), Sattur, Tamilnadu, India, kkrishnaveni@srnmcollege.ac.in*

This research proposes DeepIncepNet, a novel method that combines deep neural networks (DNNs) and hyperspectral imaging to identify illnesses in the leaves of corn or maize plants. The Specim IQ system was utilized to gather hyperspectral imaging data, which encompasses a broad range of wavelengths in spectral information. Using a unique DNN architecture, DeepIncepNet uses Inception Networks (InceptionV3) to classify healthy and damaged maize leaves. To assess the performance of the suggested model, it is compared to well-known architectures as InceptionV3, ResNet-50, and ResNet-101. The results of the experiments suggest that DeepIncepNet achieves greater robustness and accuracy in disease identification, highlighting its potential for early detection and treatment of diseases affecting the maize or corn plant.

Keywords: Corn, Maize, Hyperspectral Imaging, Deep Neural Networks, Inception Networks, DeepIncepNet, SpecimIQ, InceptionV3.

1. Introduction

One of the most significant cereal crops in the world, corn, also known as maize (Zea mays), provides millions of people with a basic diet and is essential for the production of industrial goods and animal feed. However, a number of illnesses that can drastically lower yields and the financial benefits to farmers constitute a threat to the quality and productivity of maize crops [1]. Globally, plant diseases brought on by pathogens including fungi, bacteria, viruses, and nematodes present a serious threat to systems that produce maize. These illnesses can show up as leaf spots, blights, rusts, smuts, and wilts, to name a few. They can cause symptoms including discoloration, lesions, malformations, stunted growth, and even plant mortality by affecting the leaves, stems, roots, and ears of the maize plant, among other sections of the plant. These illnesses can have a disastrous effect on agricultural productivity and quality, causing farmers to suffer significant financial losses and jeopardizing food security in many areas [2].

Effective disease management and control techniques for maize plant diseases depend on early identification and precise diagnosis. Prompt intervention strategies, like the use of fungicides, crop rotations, and the selection of resistant cultivars, can reduce the spread of disease and reduce yield losses. However, traditional disease diagnosis techniques like eye inspection and symptom-based identification are frequently laborious, prone to inaccuracy, and subjective. A rising number of people are interested in using cutting-edge technology, such deep learning and hyperspectral imaging, to identify maize plant diseases automatically and accurately in order to overcome these difficulties [3]. Spectral fingerprints from maize leaves can be captured via hyperspectral imaging at a variety of wavelengths, offering extensive data for the characterization and discriminating of diseases. Deep learning is ideally suited for image-based illness detection applications because it provides strong tools for feature extraction and pattern identification, especially with convolutional neural networks (CNNs)[4]. In this regard, the goal of this research is to create and assess a new method for detecting corn or maize plant diseases utilizing suggested DNN classifiers with Inception Networks (DeepIncepNet) and Specim IQ hyperspectral imaging. This research aims to improve the scalability, efficiency, and accuracy of disease diagnosis in maize crops by fusing cutting-edge deep learning techniques with advanced imaging technology. This will ultimately lead to better crop management practices, higher yields, and sustainable agriculture.

2. Related Works

Inception Networks (InceptionV3): Google researchers introduced Inception Networks (InceptionV3), sometimes referred to as GoogLeNet, as a deep convolutional neural network architecture for image classification applications [5]. The usage of inception modules, which are made up of several parallel convolutional layers with various kernel sizes (1x1, 3x3, and 5x5) and pooling operations, is the main novelty of Inception Networks. As a result, the network can effectively collect characteristics at various scales and resolutions. An enhanced version of Inception Networks known as InceptionV3 has been fine-tuned for more precision and effectiveness. With 48 layers, it attains cutting-edge results on picture categorization benchmarks like ImageNet. In order to minimize computational cost and increase training stability, InceptionV3 also includes additional architectural enhancements such batch normalization, factorization, and dimensionality reduction.

Residual Networks (ResNet-50, ResNet-101): Microsoft Research researchers introduced the concept of Residual Networks, or ResNets, as a way to address the issue of vanishing gradients in deep neural networks [6]. The utilization of residual connections, also known as skip links, which permit gradients to pass across the network directly without attenuation, is the fundamental principle underlying ResNets. Because these connections mitigate the vanishing gradient problem, it is possible to train very deep networks (hundreds of layers). Specific variations of Residual Networks with varying number of layers include ResNet-50 and

ResNet-101. ResNet-101 features 101 layers, compared to 50 layers in ResNet-50. Both architectures have been widely used in numerous computer vision applications and achieve state-of-the-art performance on picture categorization tasks. The idea of residual blocks—shortcut connections that eschew one or more convolutional layers—is introduced by ResNet architectures. The network can learn residual mappings thanks to these shortcut connections, which facilitates deep network optimization and improves speed.

For image classification problems, two popular deep learning architectures are Inception Networks and Residual Networks. While Residual Networks are excellent at training very deep networks by resolving the vanishing gradient issue, Inception Networks are renowned for their effectiveness at capturing multi-scale features through inception modules. Because InceptionV3 makes use of parallel convolutions and dimensionality reduction techniques, it is generally more computationally efficient than ResNet-50 and ResNet-101 [7]. Particularly when working with very deep architectures and difficult datasets, ResNet-50 and ResNet-101 may perform better on some tasks or datasets. The decision between Inception Networks and Residual Networks is influenced by various elements, including processing capacity, dataset properties, and particular performance needs for the given image classification task.

3. Hyperspectral Imaging Data Collection and Preprocessing

3.1. Specim IQ System

Because of its small size and portability, the Specim IQ system may be easily installed in a variety of field and laboratory environments. Because of its portable design, which allows for flexibility and ease of use when gathering data, it is appropriate for on-site measurements and quick evaluations [8]. High spectral resolution provided by the Specim IQ system enables thorough material characterization using their spectral signatures. It provides extensive data for research and interpretation by capturing spectral data throughout hundreds of small bands in the visible and near-infrared spectrum (400-1000 nm).

The Specim IQ system's real-time imaging and analysis capacity is one of its best features. It facilitates quick decision-making and feedback during data collection by allowing users to instantly gather hyperspectral photos and view spectral data in real-time. Multiple imaging modalities are supported by the Specim IQ system to meet the needs of various applications and users. With its snapshot and scanning modes, it can quickly image wide areas in hyperspectral mode or map specific regions of interest in precise spectral mode. Integrated light sources and calibration procedures are elements of the Specim IQ system that guarantee reliable and accurate imaging performance. It has integrated LED lighting with spectral characteristics and intensity adjustments, as well as automated sensor and spectral calibration processes [9]. For easy operation and data analysis, the Specim IQ system has an intuitive software and user-friendly interface. To adjust imaging parameters, take hyperspectral images, and carry out simple spectral analysis operations, it offers a graphical user interface (GUI). To increase its functionality and adaptability, the Specim IQ system is made to work with a variety of attachments and accessories. This consists of interchangeable filters, lenses, and adapters to accommodate various sample kinds and imaging conditions.

3.2. Hyperspectral Imaging Data Collection:

- The process of selecting an imaging system involves selecting a hyperspectral imaging system that is suitable for the study and can capture spectral information across a broad range of wavelengths, usually in the visible and near-infrared spectrum.
• Imaging Setup: Install the imaging system in a controlled space, like a field or greenhouse used to produce corn or other cereal crops. Make sure there is minimum environmental disturbance and adequate illumination.
• Image Acquisition: Using the imaging system, take hyperspectral pictures of the leaves of corn or maize plants. To capture the full leaf surface, position the camera at the proper distance and angle. Take pictures of both well and sick plants to guarantee that the degree and symptoms of the diseases vary [10].

- Calibration: To guarantee precise and reliable data across various wavelengths and imaging sessions, perform spectral and radiometric calibration of the imaging system.
3.3. Preprocessing steps including spectral calibration, noise reduction, and normalization

Hyperspectral imaging data must be preprocessed before it can be used for analysis and machine learning applications. The following describes preprocessing procedures that are especially designed for hyperspectral data obtained with the Specim IQ system, such as spectral calibration, noise reduction, and normalization:

Spectral Calibration: Accurate mapping of spectral information to matching wavelengths is ensured by spectral calibration, which accounts for variances in sensor response. Calibration of the hyperspectral data to recognized spectral standards or reference materials is the task of this stage. Using calibration targets with known spectral reflectance values, like spectralon or white reference tiles, may be necessary for spectral calibration of the Specim IQ system. By adjusting for any systematic mistakes or differences in wavelength assignment, the spectral calibration procedure matches the recorded spectral data with the anticipated spectral characteristics [11].

Noise Reduction: By reducing random noise and artifacts, noise reduction techniques seek to increase the signal-to-noise ratio of hyperspectral data. This is especially crucial for improving the accuracy and dependability of spectral signatures, especially in noisy or low-light settings. Typical methods for reducing noise in hyperspectral data are as follows:

- Averaging or smoothing: Applying spatial or spectral averaging to reduce random fluctuations and smooth out noise.

Spatial averaging:

$I_{smooth(x,y,\lambda)}=(1/N)\ \sum_{i=1}^{N} I(x,y,\lambda,i)$

Spectral averaging:

$I_{smooth(x,y,\lambda)}=(1/M)\ \sum_{j=1}^{M} I(x,y,\lambda_j)$

Where $I(x,y,\lambda)$ represents the intensity at spatial coordinates (x,y) and wavelength λ, and N is the number of neighboring pixels or spectra considered for averaging.

- Filtering: Applying spatial or spectral filters, such as median filtering or Gaussian filtering, to remove high-frequency noise while preserving important spectral features.

Median filtering:

$I_{filtered(x,y,\lambda)}$=median(I(x′,y′,λ))

Over a local neighborhood centered at (x,y).

Gaussian filtering:

$I_{filtered(x,y,\lambda)} = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} (1/2\pi\sigma^2)\, e^{(x'2+y'2)/2\sigma 2} I(x-i,y-j,\lambda)$

Where σ is the standard deviation of the Gaussian kernel.

- Spectral normalization: Dividing each spectrum by a reference spectrum or the mean spectrum to reduce spectral noise and enhance signal clarity.

$I_{normalized(x,y,\lambda)} = I(x,y,\lambda) \;/\; I_{ref}(\lambda)$

Where $I_{ref}(\lambda)$ is a reference spectrum, such as a white reference or a mean spectrum, used for normalization.

Normalization: Normalization standardizes the spectral data to account for variations in illumination conditions, sensor sensitivity, and overall intensity levels. This ensures consistency and comparability of spectral signatures across different imaging sessions and samples. Normalization techniques for hyperspectral data include:

- Min-max normalization: Scaling each spectral band to a common range (e.g., [0, 1]) based on the minimum and maximum values observed in the dataset.

$I_{normalized}(x,y,\lambda) = I(x,y,\lambda) - \min(I) \;/\; \max(I) - \min(I)$

- Z-score normalization: Standardizing each spectral band to have zero mean and unit variance, ensuring that the spectral data have a consistent distribution.

$I_{normalized}(x,y,\lambda) = (\, I(x,y,\lambda) - \mu_I \,) \;/\; \sigma_I$

Whereμ_I is the mean intensity and $\sigma_I$is the standard deviation of intensities across all spectra.

- Total intensity normalization: Scaling each spectrum by its total intensity or sum of spectral values, normalizing the overall spectral energy and intensity.

$I_{normalized}(x,y,\lambda) = I(x,y,\lambda) \;/\; \sum_{\lambda}^{I(x,y,\lambda)}$

By applying spectral calibration, noise reduction, and normalization techniques, researchers can preprocess hyperspectral imaging data collected using the Specim IQ system to improve data quality, enhance spectral features, and facilitate downstream analysis tasks such as disease detection and classification using machine learning algorithms like DeepIncepNet or ResNet.

Overview of DNN Classifiers with Inception Networks:

Using hyperspectral imaging data, the proposed deep neural network (DNN) classifiers with Inception Networks offer a novel method for disease identification in the leaves of maize or corn plants. Motivated by the remarkable outcomes of deep learning architectures, specifically Inception Networks, in image classification problems, the suggested classifiers seek to utilize the abundant spectrum data obtained by hyperspectral sensors to facilitate precise and timely disease diagnosis. The Inception architecture, which uses inception modules to extract various hierarchical features from input images, serves as the foundation for the architecture of the proposed DNN classifiers. The network's capacity to distinguish between plant tissues that are healthy and those that are ill is improved by these modules, which use parallel convolutional processes with various kernel sizes to capture features at various scales and resolutions[12].

To learn hierarchical representations of features from hyperspectral pictures, the proposed DNN classifiers with Inception Networks are made up of numerous layers of convolutional, pooling, and fully connected units arranged hierarchically. The network's layers each apply nonlinear activation functions to the input data, gradually extracting and combining variables to forecast whether or not disease will be present in the leaves of corn or maize plants.

A particular kind of artificial neural network called a convolutional neural network (CNN) is made especially for handling structured grid data, like photographs. Convolutional layers, pooling layers, fully linked layers, and activation functions are some of the layers that make it up. When it comes to image classification, a CNN uses a sequence of convolutional and pooling layers to process an input image and extract pertinent characteristics. Following that, these features are sent to fully connected layers for categorization. A particular CNN architecture called Inception V3 has been extensively utilized for a number of computer vision applications, including picture classification. The usage of inception modules, which are building elements that enable the network to record features at various spatial scales and resolutions, is what distinguishes it. The parts of Inception V3 are broken down here along with their purposes:

4. Proposed Methodology

For agricultural crops like corn or maize plants to remain healthy, maximize yields, and support food production systems, automated disease detection is essential. Conventional disease diagnosis techniques frequently rely on physical evaluation and visual inspection, which can be error-prone, labor-intensive, and subjective. This study uses hyperspectral imaging and deep learning classifiers based on Inception Networks (DeepIncepNet) to offer a unique way for automated disease identification in corn or maize plant leaves in order to overcome these issues. Across a broad range of wavelengths, hyperspectral imaging provides a potent tool for extracting comprehensive spectral information from plant tissues. Hyperspectral imaging allows the identification of minor changes associated with disease signs, such as discolouration, necrosis, and physiological stress responses, by examining the distinct spectral fingerprints of healthy and diseased plants. The goal of this work is to provide a reliable system for the early and accurate diagnosis of illnesses in corn or maize plants by utilizing the spectral richness and spatial resolution of hyperspectral data.

A number of crucial processes are included in the suggested methodology, such as data collection, preprocessing, feature extraction, model creation, assessment, and validation. Specim IQ system-collected hyperspectral imaging data are preprocessed to minimize noise, adjust for sensor calibration errors, and standardize spectral data. From the preprocessed data, pertinent spectral and spatial characteristics are retrieved and fed into deep learning classifiers, such as DeepIncepNet.

DeepIncepNet Architecture for Disease Detection:

With an emphasis on corn or maize plants, DeepIncepNet is a deep learning architecture created especially for automated disease detection in agricultural crops. DeepIncepNet, which is adapted to hyperspectral imaging data and inspired by the Inception Networks (InceptionV3) architecture, seeks to use the rich spectral information obtained by hyperspectral sensors for precise and timely disease detection in plant leaves. As seen in Figure 1, DeepIncepNet's architecture is based on the concepts of convolutional neural networks (CNNs) and incorporates many of the essential elements of the Inception architecture, which is well-known for being highly efficient in image classification tasks.

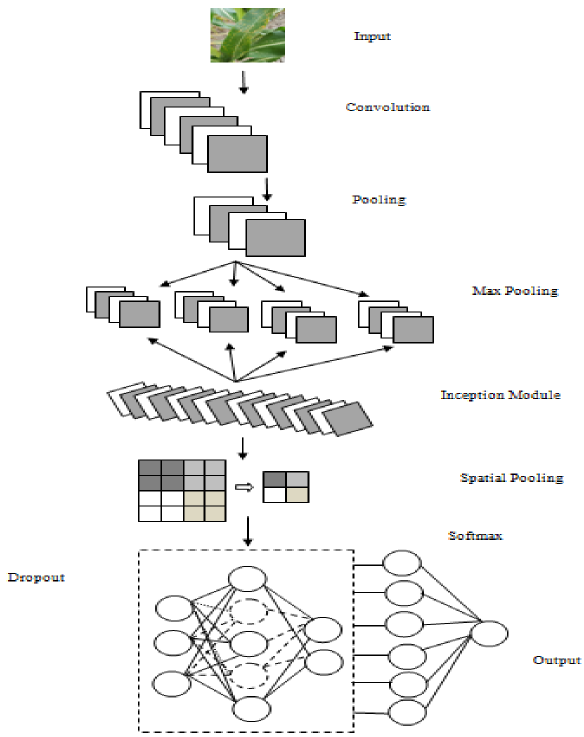

Figure 1: Architecture of DeepIncepNet

DeepIncepNet extracts hierarchical feature representations from hyperspectral pictures by arranging layers upon layers of convolutional, pooling, and fully connected units. Inception modules, which act as the building blocks for feature extraction, are the essential components of DeepIncepNet. By combining concurrent convolutional procedures with various kernel sizes (1x1, 3x3, and 5x5), these modules enable the network to collect features at various spatial resolutions and scales. DeepIncepNet improves its ability to detect diseases by learning to extract a variety of discriminative features from hyperspectral data by merging feature maps from various neural paths inside each inception module. Moreover, DeepIncepNet integrates extra architectural improvements to conform to hyperspectral imaging data properties. The network may concentrate on pertinent spectrum properties while lowering computational complexity thanks to spectral pooling layers, which combine spectral data from several bands.

Moreover, DeepIncepNet has spectral normalization layers to improve the network's resistance to changes in light and sensor sensitivity. In order to train DeepIncepNet, its parameters must be optimized using labeled hyperspectral data, in which each sample has a label indicating whether or not disease is present in the leaves of corn or maize plants. By using stochastic gradient descent optimization and backpropagation to minimize a predetermined loss function, the network gains the ability to distinguish between samples that are healthy and those that are sick.

Several essential elements make up the architecture of DeepIncepNet, a deep learning classifier based on Inception Networks designed for disease identification in corn or maize plant leaves utilizing hyperspectral imaging data. Let's examine DeepIncepNet's intricate architecture and its constituent parts:

Input Image: Hyperspectral images of the leaves of corn or maize plants are sent into the input layer of DeepIncepNet. In the hyperspectral image, every pixel represents a spectrum of reflectance values spanning several wavelength bands. Assume that the input image is a 299x299x3 array of pixel values that correspond to the image's RGB colors (height, width, and channels).

Convolutional Layers: Using learnable filters, multiple convolutional layers in DeepIncepNet convolve input hyperspectral pictures to extract spectral and spatial characteristics. Local structures and patterns in the hyperspectral data are captured by these convolutional layers. In order to extract information from the input image, such as edges, textures, and patterns, these layers use convolutional filters [13]. Throughout the training phase, the filter weights are discovered. The input image is first processed through a number of convolutional filters by Inception V3. These filters are tiny matrices, such as 3x3 or 5x5, that are dragged over the image to identify characteristics such as contours, textures, and forms. A convolutional filter, for instance, could be able to identify horizontal lines in the picture. Gradients or diagonal edges might be picked up by another filter.

Convolutional layer in a convolutional neural network (CNN) involves applying a convolution operation between the input feature map and a set of learnable filters (also known as kernels).

Input feature map I of size $H \times W \times C_{in}$, where H is the height, W is the width, and C_{in} is the number of input channels.

Learnable filters F of size $F_h \times F_w \times C_{in} \times C_{out}$, where F_h is the filter height, F_w is the filter width, C_{out} is the number of output channels.

The formula for computing the output feature map O of the convolutional layer is:

$$O_{i,j,k}=\sum_{l=1}^{Cin}\sum_{m=1}^{Fh}\sum_{n=1}^{Fw} I_{i+m-1,j+n-1,l}\times F_{m,n,l,k}$$

Where, $O_{i,j,k}$ is the value at position (i,j) of the output feature map in the k^{th} channel. $I_{i+m-1,j+n-1,l}$ is the value at position (i+m−1,j+n−1) of the input feature map in the l^{th} channel. $F_{m,n,l,k}$ is the value of the filter at position (m,n) in the l_{th} input channel and k_{th} output channel.Additionally, a bias term may be added to each output channel, resulting in the following modified formula:

$$O_{i,j,k}=\sum_{l=1}^{Cin}\sum_{m=1}^{Fh}\sum_{n=1}^{Fw} I_{i+m-1,j+n-1,l}\times F_{m,n,l,k}+B_k$$

Where, B_k is the bias term for the k^{th} output channel.This bias term allows the network to learn an offset for each filter, providing additional flexibility in modeling complex relationships in the data.

Pooling Layers: The feature maps that are acquired from the convolutional layers are downsampled using pooling layers, which lowers their spatial dimensions without losing any significant information. Two popular pooling operations are max pooling and average pooling. The feature maps are downsampled using max pooling or average pooling after each convolutional layer. Pooling preserves crucial information while assisting in reducing the feature maps' spatial dimensions. By taking the maximum value inside each pooling zone, for instance, max pooling may efficiently reduce the size of the feature maps.Max pooling and average pooling are the two most popular pooling techniques. The following are the two formulas:

Max Pooling:Max pooling takes the maximum value within each pooling region. Given an input feature map I of size H×W×C and a pooling window size of P×P, the output feature map O is computed as follows:

$$O_{i,j,k}=\max_{m=1}^{P}\max_{n=1}^{P} I(i-1)_{P+m},(j-1)_{P+n,k}$$

Where,$O_{i,j,k}$ is the value at position (i,j) of the output feature map in the k^{th} channel.$I(i-1)_{P+m},(j-1)_{P+n},k$ is the value at position $(i-1)_{P+m},(j-1)_{P+n}$ of the input feature map in the k^{th} channel.

Average Pooling:Average pooling computes the average value within each pooling region. Given an input feature map I of size H×W×C and a pooling window size of P×P, the output feature map O is computed as follows:

$$O_{i,j,k}=(1/P^2)\sum_{m=1}^{P}\sum_{n=1}^{P} I(i-1)_{P+m},(j-1)_{P+n,k}$$

Where, $O_{i,j,k}$ is the value at position (i,j) of the output feature map in the k^{th} channel.$I(i-1)_{P+m},(j-1)_{P+n,k}$ is the value at position $(i-1)_{P+m},(j-1)_{P+n}$ of the input feature map in the k^{th} channel.These pooling operations help reduce the spatial dimensions of the feature maps, making them more computationally efficient to process while preserving important spatial information. Additionally, pooling helps in achieving translational invariance, making the network more robust to small variations in the input data.

Spectral pooling layers: Spectral pooling layers in DeepIncepNet are designed to aggregate spectral information across multiple bands while reducing the dimensionality of the spectral data. The goal is to preserve relevant spectral features, enhance computational efficiency, and reduce overfitting. Here's the conceptual formula for spectral pooling layers:

- Input hyperspectral data X of size H×W×B, where H is the height, W is the width, and B is the number of spectral bands.
- Spectral pooling window size P (typically along the spectral dimension).

The spectral pooling operation aggregates spectral information within each pooling window along the spectral dimension. Here's how the spectral pooling layer operates:

$$Y_{i,j,k}=\text{Pooling}(X_{i,j,k-P/2},X_{i,j,k-P/2+1},\ldots,X_{i,j,k+P/2})$$

Where, $Y_{i,j,k}$ is the output value at position (i,j) in the pooled feature map along the k^{th} spectral band.Pooling is the pooling operation applied along the spectral dimension within the pooling window. This operation could be average pooling, max pooling, or other aggregation techniques.$X_{i,j,k-P/2}, X_{i,j,k-P/2+1}, \ldots, X_{i,j,k+P/2}$ represent the spectral values within the pooling window centered at position (i,j,k) in the input hyperspectral data X.

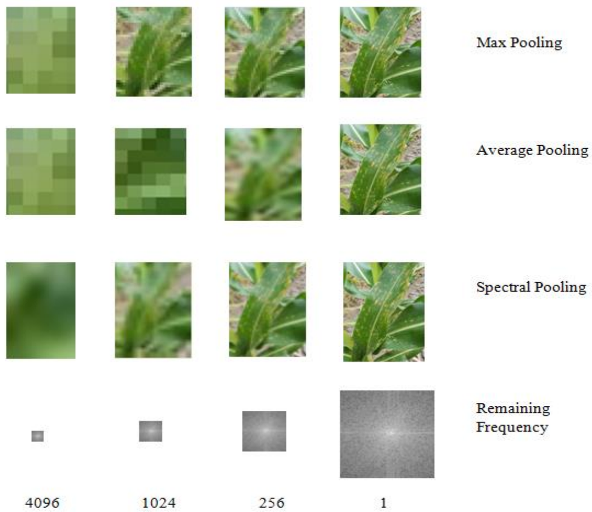

Figure 2: Approximations for different pooling schemes, for different factors of dimensionality reduction

Spectral pooling saves far more information and structures for the same number of parameters, as shown in the third row of Figure 2. This is due to the fact that the spectral transform provides a sparse basis in the frequency domain frameworks for the inputs including spatial components. The spectrum power of an input is usually concentrated in the lower frequencies, with the higher frequencies acting mainly as noise encoders. This non-uniformity in spectrum intensity allows for the removal of high frequencies with minimal damage to the input data. By combining spectrum data from each pooling window, the spectral pooling layer efficiently reduces the number of spectral bands from B to a manageable number while maintaining pertinent spectral properties. This dimensionality reduction improves computing performance and lowers the possibility of overfitting, particularly when noisy or redundant bands are included in the original spectrum data.

The unique needs of the application and the properties of the hyperspectral data will determine which pooling operation (such as average or maximum pooling) and pooling window size P are used.

Spectral normalization layers: Spectral normalization layers in DeepIncepNet are intended to enhance the network's robustness to variations in illumination conditions and sensor sensitivity by normalizing spectral data. The normalization process ensures consistent performance across different imaging conditions. Here's the conceptual formula for spectral normalization layers:

- Input hyperspectral data X of size H×W×B, where H is the height, W is the width, and B is the number of spectral bands.

The spectral normalization operation normalizes the spectral values within each band to account for variations in intensity and spectral response. Here's how the spectral normalization layer operates:

$Y_{i,j,k}=(X_{i,j,k}-\mu_k) / \sigma_k$

Whereas Yi,j,k represents the normalized value at location (i,j) in the output hyperspectral data's spectral band k. The spectral value at point (i,j) in spectral band k of the input hyperspectral data X is denoted by the notation Xi,j,k.The average spectral value for every pixel in band k is denoted by μk. The standard deviation of spectral values for every pixel in band k is denoted by σk. Each spectral value is normalized by deducting the mean (muk) from it and dividing the result by the standard deviation (sigmak). This process effectively normalizes the spectral data by guaranteeing that the spectral values have zero mean and unit variance within each band. By ensuring that the network receives consistent input data regardless of the imaging settings, spectral normalization layers help reduce the effects of differences in illumination conditions and sensor sensitivity. This normalization improves the network's generalization performance across various contexts and imaging situations and increases its capacity to extract relevant features from the spectral data.

Inception Modules: Inspired by the Inception design, the fundamental building pieces of DeepIncepNet are called inception modules. Parallel convolutional operations with various kernel sizes (1x1, 3x3, and 5x5) and pooling operations make up each inception module. The network is able to extract a variety of distinct and discriminative properties from hyperspectral data because to these parallel paths that capture features at various spatial scales and resolutions. The main new feature of Inception V3 is Inception modules. They enable the network to capture features at various scales and resolutions. They are composed of numerous parallel convolutional processes with varied kernel sizes. This aids in reducing computing complexity and enhancing the network's representational power. Inception V3 captures features at various resolutions and scales by using inception modules. Different kernel sizes are used in parallel convolutional procedures to create each inception module. An inception module, for instance, may include parallel routes with convolutions of 1x1, 3x3, and 5x5. These paths record characteristics at various abstraction levels, ranging from minute details to more extensive patterns.

Input feature map I of size $H \times W \times C_{in}$, where H is the height, W is the width, and C_{in} is the number of input channels.

Learnable filters for each pathway: F^1, F^3, F^5, and optionally F^{reduce} for dimensionality reduction.

The output of the Inception module O is computed as follows:

$O=Concat(O^1,O^3,O^5,O^{reduce})$

Where, O^1 is the output of the 1x1 convolution pathway.O^3 is the output of the 3x3 convolution pathway.O^5 is the output of the 5x5 convolution pathway.O^{reduce} is the output of the dimensionality reduction pathway (if present).Concat denotes the concatenation operation along the channel dimension.

The individual outputs O^1, O^3, O^5, and O^{reduce} are computed as follows:

- 1x1 Convolution Pathway (Optional): This pathway applies 1x1 convolutional filters to the input feature maps. These filters capture local correlations and interactions within the feature maps.

$O^1=ReLU(I*F^1)$

- 3x3 Convolution Pathway: This pathway applies 3x3 convolutional filters to capture features at a slightly larger spatial scale compared to the 1x1 pathway Figure 3. The larger receptive field of these filters allows them to capture more complex patterns and structures.

$O^3=ReLU(I*F^3)$

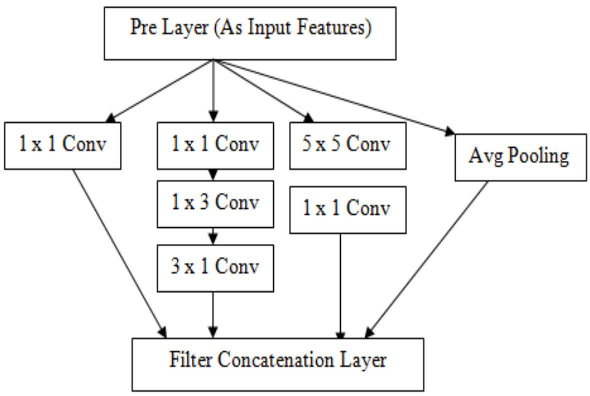

Figure 3: 3x 3 Convolution Pathways

- 5x5 Convolution Pathway: This pathway applies 5x5 convolutional filters to capture features at an even larger spatial scale Figure 4. These filters are capable of capturing global patterns and structures within the input feature maps.

$O^5=ReLU(I*F^5)$

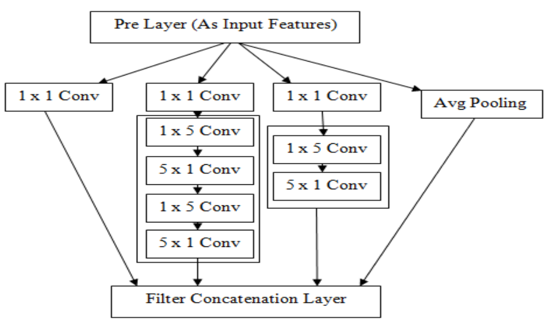

Figure 4: 5 x 5 Convolution Pathways

- Dimensionality Reduction Pathway (Optional):

To reduce the computational cost and prevent the explosion of the number of parameters, Inception modules often include 1x1 convolutional layers before the larger convolutional operations. These 1x1 convolutions act as dimensionality reduction steps by reducing the number of input channels before applying the larger convolutions.

$O^{reduce}=ReLU(I*F^{reduce})$

* denotes the convolution operation.ReLU denotes the Rectified Linear Unit activation function applied element-wise.The output size of each pathway depends on the size of the input feature map and the dimensions of the learnable filters.

Concatenation Layer: The feature maps acquired from several parallel convolutional pathways are concatenated by inception modules. The network may include many features that were retrieved at various scales and resolutions thanks to this concatenation. Along the channel dimension, the output feature maps from the various paths in the inception module are concatenated.For example, the concatenated feature map would be 100x100x(64+64+64)=100x100x192 if the 1x1, 3x3, and 5x5 paths each yield feature maps of size 100x100x64.

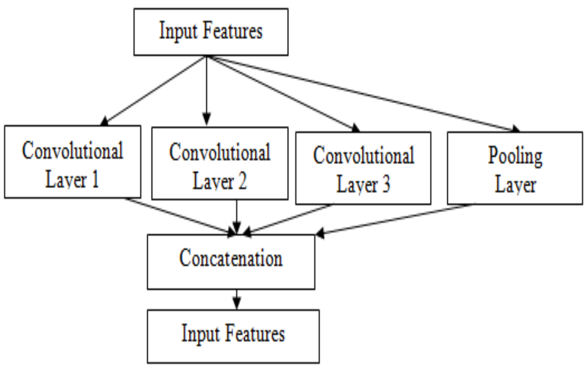

Figure 5: Convolutional pathways

The concatenation layer in a neural network combines the outputs of multiple pathways or branches into a single output tensor by stacking them along a specified axis, typically the channel dimension Figure 5. The concatenation operation is given:

- Output tensors $O^1,O^2,\ldots,O^N$ from O^N different pathways.

- Each output tensor O^i has the same spatial dimensions H×W but possibly different numbers of channels C_i.

The concatenated output tensor O_{concat} is computed as follows:

$O_{concat}=Concat(O^1,O^2,\ldots,O^N)$

Where ,O_{concat} is the concatenated output tensor.Concat denotes the concatenation operation.

The concatenation operation stacks the output tensors along the channel dimension. If O^i has shape $H \times W \times C_i$, then the concatenated output tensor O_{concat} will have shape $H \times W \times (C_1 + C_2 + \ldots + C_N)$.

In mathematical terms, the concatenation operation can be represented as follows:

$$O_{concat}[i,j,:] = \begin{cases} O^1[i,j,:], \text{ if } 0 \leq i < H, 0 \leq j < W, \\ O^2[i,j,:], \text{ if } H \leq i < 2H, 0 \leq j < W, \\ O^N[i,j,:], \text{if } (N-1)H \leq i < NH, 0 \leq j < W \end{cases}$$

In this expression:

- $O_{concat}[i,j,:]$ denotes the (i,j)th row and all channels of the concatenated output tensor.
- The rows of the concatenated tensor are filled with the corresponding rows from the output tensors of each pathway.

The concatenation layer is commonly used in neural network architectures, including multi-branch architectures such as the Inception modules in InceptionNet, to combine features extracted from different pathways before passing them to subsequent layers for further processing.

Fully Connected Layers: At the end of the network, DeepIncepNet has completely connected layers that enable categorization using the features that have been extracted. The high-dimensional feature representations are converted into probability distributions over various disease or health condition classes by these layers.One or more fully connected layers pass across the flattened feature maps that the convolutional layers produced. By developing intricate mappings between the retrieved features and the output classes, these layers carry out classification. After being combined into a vector, the concatenated feature maps are run through one or more completely connected layers. Dense layers, or fully linked layers, are frequently found in neural network topologies. Every neuron in a completely connected layer is connected to every other neuron in the layer before it, creating a fully connected graph structure [14]. The flattened vector, for instance, might be linked to two completely connected layers: one with 1,024 neurons and the other with 512 neurons. The following formula can be used to determine a completely connected layer's output:

Given:

- Input vector X of size M x 1,where M is the number of neurons in the preceding layer.
- Weight matrix W of size N xM,where N is the number of neurons in the current layer.
- Bias vector b ofsize Nx 1,where N is the number of neurons in the current layer.

The output vector Y of the fully connected layer is computed as follows:

$$Y = W.X + b$$

where (.) indicates the matrix multiplication operation and Y is the output vector of size $N \times 1$.The matrix multiplication of the input vector X and the weight matrix W is denoted by

W.X. The weighted total of the inputs for every neuron in the current layer is calculated by this process. A bias term is added element-wise for each neuron by adding b to the matrix multiplication result. The network is able to determine an offset for every neuron, regardless of the input, thanks to this bias term. The fully linked layer adds bias terms element-by-element after completing a linear transformation. The Rectified Linear Unit (ReLU) or the softmax function are two examples of activation functions that are used to pass the output of the fully connected layer through in order to introduce non-linearity and allow the network to learn complex relationships in the data.In neural network architectures, fully connected layers are frequently employed for tasks like classification and regression, where the objective is to learn mappings between input features and output labels or predictions.

Dropout: In order to avoid overfitting, dropout is a regularization technique that CNNs frequently employ. During training, a predetermined percentage of neurons are randomly removed, pushing the network to acquire more resilient and universal properties. In order to avoid overfitting, dropout is applied to the output of fully linked layers during training. A predetermined percentage of neurons are dropped at random.In dropout, a random subset of neurons in the layer is "dropped out" for a brief period of time during training, which results in zero contributions from those neurons to the subsequent layer. During training, dropout may, for instance, arbitrarily set 20% of the neurons in the fully connected layer to zero.Here's the formula for dropout:

- Input vector X of size $M\times1$, where M is the number of neurons in the layer.
- Dropout probability p, which represents the probability of dropping out a neuron during training.

During training:

- Each neuron's output is multiplied by a dropout mask D of the same size as X. The dropout mask is a binary mask where each entry is set to 0 with probability p and 1 with probability 1 - p. The dropout mask D is randomly generated for each mini-batch.
- The output vector Y of the dropout layer is computed as follows:

$$Y=D\odot X$$

Where, Y is the output vector of size M x 1. $\odot$ denotes element-wise multiplication (Hadamard product).

During inference (testing):

- No dropout is applied during inference. Instead, the output Y is scaled by 1 - p to compensate for the dropout applied during training:

$$Y=(1-p).\ X$$

- The scaling factor 1 - p ensures that the expected value of Y remains the same during training and inference, helping to maintain consistency in the model's behavior.

In order to allow the activations to be regularized instead of the raw inputs, dropout is usually performed after the activation function in each layer. Dropout helps prevent the network from relying too much on any one neuron by randomly removing neurons during training, which forces the network to learn more resilient and universal properties. As a result, overfitting is decreased and the model's capacity to generalize to new data is enhanced.

Activation Functions: After each convolutional and fully connected layer, nonlinear activation functions, such as sigmoid or ReLU (Rectified Linear Unit), are added to induce nonlinearity and allow the network to learn intricate mappings between input and output. Neural networks rely heavily on activation functions to introduce nonlinearity, which helps the network recognize intricate patterns and relationships in the input. Let's examine two popular activation functions, sigmoid and ReLU (Rectified Linear Unit), and some instances of how they are used.

ReLU (Rectified Linear Unit): ReLU is one of the most widely used activation functions due to its simplicity and effectiveness. It outputs the input value if it is positive, and zero otherwise. Mathematically, the ReLU function can be defined as: $f(x)=\max(0,x)$

Example: Let's consider an input value $x=2$.

The output of the ReLU function would be:

$f(2)=\max(0,2)=2$

Similarly, for an input value $x=-1$, the output of the ReLU function would be:

$f(-1)=\max(0,-1)=0$

ReLU effectively introduces sparsity in the network, as it turns off neurons that receive negative inputs, which helps in preventing the vanishing gradient problem during training.

Sigmoid: The sigmoid function is commonly used in binary classification problems where the goal is to predict probabilities. It squashes the input values to the range between 0 and 1, allowing them to be interpreted as probabilities. Mathematically, the sigmoid function is defined as:

$f(x)= 1 / (1+e^{-x}1)$

Example: Let's consider an input value $x=1$. The output of the sigmoid function would be:

$f(1)= 1 /(1+e^{-1})\approx 0.731$

Similarly, for an input value $x=-2$, the output of the sigmoid function would be:

$f(-2)= 1 / (1+e^{2}) \approx 0.119$

Any real-valued input can be mapped to the interval (0, 1) by the sigmoid function, which makes it appropriate for tasks like binary classification where the output must be read as a probability. ReLU and sigmoid activation functions give the neural network nonlinearity, which enables it to recognize intricate links in the data and produce predictions that are more accurate. Usually, they are applied to the output of each neuron in the network, element by element.

Softmax Function: The output of the last fully connected layer is subjected to the softmax function in order to transform the raw scores into class probabilities. The probability of an input image falling into a specific class is represented by each value. A vector of raw scores is produced by the last fully connected layer, and they are then run through the softmax function to determine the class probabilities. The softmax function, for instance, would transform the raw scores into probabilities showing the chance that the input image belongs to each of the ten classes (dog, cat, bird, etc.). at neural networks, the softmax function is frequently used to transform raw scores or logits into probabilities, especially at the output layer for multi-class classification problems. The softmax function takes a vector of real-valued numbers as input and outputs a probability distribution over multiple classes. Here's the formula for the softmax function:

- Input vector Z of size K×1, where K is the number of classes.
- Element z_i in Z represents the raw score or logit for class i.

The softmax function computes the probability y^Iof class i as follows:

$\hat{y}_i = (e^{z_i}) / \sum_{j=1}^{K} e^{z_j}$

Where,e denotes the base of the natural logarithm (Euler's number). $\sum_{j=1}^{K} e^{z_j}$represents the sum of the exponentiated raw scores over all classes, ensuring that the resulting probabilities sum to 1.

- e^{z_i}computes the exponential of the raw score z_i for class i , transforming it into a positive value.
- The exponential function emphasizes larger raw scores, amplifying the differences between scores and making them more distinguishable.
- Dividing each exponentiated raw score by the sum of all exponentiated scores normalizes the values, ensuring that the resulting probabilities are between 0 and 1 and sum to 1.

Higher raw scores are correlated with higher probabilities in a probability distribution, which is effectively created by the softmax function. It makes it possible for the model to produce probabilities that show how likely it is for each class given the input.The softmax function is commonly used in conjunction with a loss function, such as cross-entropy loss, during training in order to calculate the loss and backpropagately update the model parameters. The class with the highest probability output by the softmax function is typically predicted as the final class label during inference (testing).

Loss Function: Using a suitable loss function, like cross-entropy loss, DeepIncepNet is optimized during training to minimize the difference between predicted and ground-truth labels in the training dataset. Gradient descent and backpropagation are used to iteratively update the network's parameters. A loss function is necessary to measure the difference between the model's predicted outputs and the ground-truth labels in the training dataset during the DeepIncepNet training process. The particular job being solved—such as regression, classification, or other tasks—determines the loss function to be used. This section will concentrate on the popular cross-entropy loss function for classification applications.

Cross-Entropy Loss Function: Cross-entropy loss is commonly used in classification problems, where the goal is to predict the probability distribution of class labels. It measures the dissimilarity between the predicted probability distribution and the actual distribution of class labels. For a multi-class classification problem with N classes, the cross-entropy loss between the predicted probabilities $y^{\wedge}$and the ground-truth labels y can be defined as:

CE(y^,y)=−N1∑i=1Nyilog(y^i)

Where, $y^{\wedge}$ is the predicted probability distribution over the N classes. y is the ground-truth label distribution (one-hot encoded). y^{i} and $y^{\wedge i}$ are the i^{th} elements of the ground-truth and predicted distributions, respectively.

The cross-entropy loss penalizes the model more severely when it makes confident incorrect predictions, leading to more effective learning.

Training Process: The objective of the training procedure is to minimize the cross-entropy loss by utilizing optimization methods like stochastic gradient descent (SGD), Adam, or RMSprop to modify the network's parameters (weights and biases). The model parameters are iteratively updated to minimize the loss by using backpropagation to determine the gradients of the loss function with respect to them. The model learns to generate better predictions and to generalize well to new data by reducing the loss function. The DeepIncepNet training process is guided by the loss function, which measures the difference between the predicted and ground-truth labels. The optimization algorithms then modify the network's parameters to reduce this difference. This iterative process continues until the model converges to a state where the loss is minimized and the model makes accurate predictions.

DeepIncepNet Algorithms

DeepIncepNet involves detailing the steps of forward propagation (inference) and backpropagation (training).

Forward Propagation (Inference):

Input: Hyperspectral image X of size H×W×C, where H is height, W is width, and C is the number of spectral bands.

Initialization:

Initialize input as A[0]=X.

Convolutional Layers:

For each convolutional layer l with parameters W[l] (weights) and b[l] (biases):

$Z^{[l]}=A^{[l-1]} * W^{[l]}+b^{[l]}$

$A^{[l]}=\mathrm{ReLU}(Z^{[l]})$

Where* denotes the convolution operation and ReLU is the rectified linear activation function.

Inception Modules:

For each inception module:Concatenate feature maps from parallel convolutional pathways.

Spectral Pooling Layers:Apply spectral pooling to aggregate spectral information across bands.

Fully Connected Layers:Flatten the feature maps A[l] into a vector Aflat[l].or each fully connected layer l with parameters W[l] and b[l]:

$Z^{[l]}=W^{[l]}.A_{flat}^{[l]}+b^{[l]}$

$A^{[l]}=ReLU(Z^{[l]})$

Output Layer:Compute the raw scores Z[L] using the last fully connected layer.Apply softmax activation to obtain class probabilities:

$Y^{\wedge}=softmax(Z^{[L]})$

Output: Predicted class probabilities $Y^{\wedge}$.

Backpropagation (Training):

Compute Loss: Using cross-entropy loss between predicted probabilities Y^ and ground truth labels Y:

$L=-(m/1) \sum_{i=1}^{m}\sum_{c=1}^{C}Y_{ic}\log(Y^{\wedge}_{ic})$

Where m is the number of samples, C is the number of classes, Y_{ic} is the indicator function for the true class of sample i, and $Y^{\wedge}_{ic}$ is the predicted probability for class c of sample i.

Backpropagate Gradients:Compute the gradient of the loss with respect to the parameters of the network using backpropagation.

Update Parameters:Update the parameters $W^{[l]}$ and $b^{[l]}$ using gradient descent:

$W^{[l]}=W^{[l]}-\alpha. (\partial L / \partial W^{[l]})$

$b^{[l]}=b^{[l]}-\alpha. (\partial L / \partial b^{[l]})$

Where α is the learning rate.These algorithms detail the steps involved in both forward propagation (inference) and backpropagation (training) of DeepIncepNet, enabling the network to make predictions and update its parameters iteratively during training.

5. Experimental Results

Dataset: The objective of the training procedure is to minimize the cross-entropy loss by utilizing optimization methods like stochastic gradient descent (SGD), Adam, or RMSprop to modify the network's parameters (weights and biases). The model parameters are iteratively updated to minimize the loss by using backpropagation to determine the gradients of the loss function with respect to them. The model learns to generate better predictions and to generalize well to new data by reducing the loss function. The DeepIncepNet training process is guided by the loss function, which measures the difference between the predicted and ground-truth labels. The network's parameters are subsequently adjusted by the optimization algorithms to lessen this discrepancy. Three sets of the dataset were randomly selected: seventy percent of the patches were included in the training set, twenty percent were included in the validation set, and ten percent were included in the test set. There were two types of splitting methods defined: patchwise splitting, which is straightforward, and tilewise splitting, which is hard. While all of the patches from a tile must be a part of the same set in the hard split, the easy split allows patches from the same tile to appear in separate sets. Here, four type of infected leaf was considered for this experiment as sample.

1) Common rust(Pucciniasorghi) appears as small, circular to oval-shaped pustules on the upper surface of corn leaves. Figure 6 (a), these pustules are typically orange to reddish-brown in color and may occur singly or in clusters. As the infection progresses, the pustules may coalesce, leading to larger areas of discoloration on the leaves. Severe infections of common rust can reduce the photosynthetic capacity of the affected leaves, leading to reduced plant vigor and yield loss [15].

2) Corn Leaf Blight (Cochliobolusheterostrophus) manifests as large, elongated lesions with tan centers and dark-brown to reddish-brown borders on corn leaves. These lesions may coalesce, leading to extensive blighting of the foliage. Figure 6 (b), the disease is favored by warm, humid conditions. Severe infections of southern corn leaf blight can cause significant yield losses by reducing the photosynthetic area of the leaves and weakening the plants, making them more susceptible to lodging.

3) Northern Corn Leaf Blight (Exserohilumturcicum) Large, elliptical lesions with tan centers and dark-brown to reddish-brown borders develop on corn leaves. Lesions may have a water-soaked appearance and may be surrounded by a yellow halo.Figure 6 (c),Northern corn leaf blight can cause significant yield losses by reducing the photosynthetic area of leaves and weakening plants, making them more susceptible to lodging.

4) Gray Leaf Spot (Cercosporazeae-maydis) Small, rectangular lesions with yellow halos develop on corn leaves, initially appearing grayish-green and later turning grayish-brown. Lesions may have a "frogeye" appearance. Figure 6 (d), Gray leaf spot can lead to decreased photosynthetic activity in infected leaves, reducing plant vigor and potentially causing yield losses if not controlled [16].

(a) (b) (c) (d)

Figure 6 : a) Infected Image leaf 1(Common rust (Pucciniasorghi) b) Infected Image leaf 2 - Corn Leaf Blight (Cochliobolusheterostrophus) c) Infected Image 3- Northern Corn Leaf Blight (Exserohilumturcicum) d) Infected Image 4-Gray Leaf Spot (Cercosporazeae-maydis)

Performance Evaluation: All experiments were conducted using TNCornNet on a system equipped with an Intel Core-i7 processor, 8 GB of RAM, a 64-bit Windows 10 operating system, and a 512 GB SSD. Network training environments: CUDA10.2, Python 3.8.5, Tensorflow-gpu2.2.0. A training-to-testing ratio of 70:30 was employed for each experiment. This section provides a comprehensive analysis of the results, along with details on the evaluation metrics utilized. The primary metric for assessing classification performance is the classification accuracy, which is defined as the ratio of correctly classified instances to the total number of instances in the dataset. Mathematically, it is expressed as:

Accuracy = (tp + tn) / (tp + tn + fp + fn) ---(1)

Where tp denotes true positives, tn denotes true negatives, fp denotes false positives, and fn denotes false negatives.

Recall (R) and precision (P) are also commonly used metrics in image classification evaluation. Precision represents the proportion of accurately classified instances to all instances classified as positive:

Precision(P) = tp/(tp + fp) --- (2)

Recall, on the other hand, measures the proportion of accurately classified instances to all instances that should have been classified as positive:

Recall (R) = tp/(tp + fn) ----(3)

The F-score, a metric that combines precision and recall into a single value, is used to evaluate the overall performance of the system. It is computed as the harmonic mean of precision and recall:

F- Score = 2 × (P.R)/ (P + R) ---- (4)

A higher F-score indicates better predictive capability of the system. The F-score is particularly useful when comparing performance across different strategies, especially in cases where one strategy outperforms another but has a lower recall rate.

We assess the performance of many deep learning models, such as InceptionV3, ResNet-50, ResNet-101, and the suggested DeepIncepNet, in this experiment when it comes to categorizing photos of diseased corn plant leaves. Images of two common infections—Pucciniasorghi, or common rust, and Cochliobolusheterostrophus, or southern corn leaf blight—make up the benchmark dataset. In order to evaluate each model's ability to correctly identify the type of infection present in the leaves of maize plants, it is trained and tested on this dataset. The models are initialized with pre-trained weights and then refined with the use of the TNCornNet dataset to conform to the unique features of photos of infected corn plant leaves. InceptionV3's classification accuracy for photos of diseased corn plant leaves is evaluated, offering information about how useful this model architecture is for the given task. In a similar manner, InceptionV3's performance and ResNet-50's classification accuracy for photos of diseased corn plant leaves are assessed and contrasted. In comparison to InceptionV3 and ResNet-50, the accuracy of ResNet-101 is evaluated in order to determine how a deeper architecture affects classification performance. The suggested DeepIncepNet model, which incorporates inception modules into a unique architecture, is assessed for accuracy. As a result, DeepIncepNet and the baseline models (InceptionV3, ResNet-50, and ResNet-101) may be directly compared.

The proposed DeepIncepNet architecture achieves superior accuracy in classifying images of infected corn plant leaves compared to other models like InceptionV3, ResNet-50, and ResNet-101 for several technical reasons. Firstly, DeepIncepNet incorporates inception modules, which efficiently capture multi-scale features within the network. This allows the model to extract intricate patterns and details present in the images, enhancing its ability to discriminate between different types of infections. Secondly, DeepIncepNet's custom architecture optimization tailors the model specifically for the task at hand, as it is trained from scratch on the TNCornNet dataset. This customization enables DeepIncepNet to learn features optimized for the dataset's characteristics, contributing to improved classification accuracy. Additionally, DeepIncepNet benefits from adaptive learning during training, adjusting its parameters based on the dataset's gradients. This adaptability allows the model to better capture subtle variations in the images, leading to more discriminative features and higher accuracy in classification.

Furthermore, DeepIncepNet's use of dropout layers promotes the learning of more resilient and broadly applicable features and helps avoid overfitting. Dropout enhances DeepIncepNet's capacity to generalize to new data and perform better in disease classification tasks by efficiently regularizing the learning process. Finally, DeepIncepNet leverages the benefits of both conventional convolutional layers and inception modules to capture a wider variety of features and arrive at a more thorough comprehension of the intricate visual patterns found in the images. Figure 7 and Table 1 Shown, these technical design choices and optimizations collectively contribute to DeepIncepNet's superior accuracy in classifying images of infected corn plant leaves compared to other models.

Table 1: Classification accuracy for TNCornNet (Sample 4) image benchmark.

Model	Classification Accuracy (%)			
	Infected Image1 (Pucciniasorghi)	Infected Image2 (Cochliobolusheterostrophus)	Infected Image 3(Exserohilumturcicum)	Infected Image 4(Cercosporazeae-maydis)
ResNet50	91.25	92.54	90.72	91.44
ResNet101	92.41	92.89	91.99	92.19
InceptionV3	95.41	96.47	95.88	96.77
DeepIncepNet	97.78	98.65	97.35	98.73

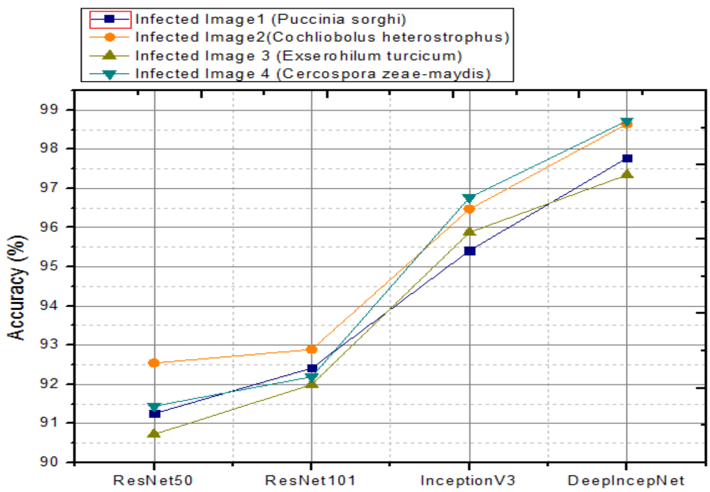

Figure 7: Classification Accuracy for TNCornNet (Sample 4) image benchmark.

The F-score, which represents the harmonic mean of precision and recall, is calculated for each model. It provides a comprehensive measure of the models' overall performance in disease classification, taking into account both true positive and false positive rates. DeepIncepNet may achieve the best F-score due to its ability to effectively balance precision and recall, capturing a high proportion of true positives while minimizing false positives and false negatives. Recall measures the proportion of correctly identified instances of a class to all instances belonging to that class. DeepIncepNet may achieve higher recall compared to other models by effectively capturing a greater proportion of true positive instances of infected corn plant leaves, leading to fewer false negatives.

Model	Precision (%)	Recall (%)	F-Score (%)
Infected Image 1(Puccinia sorghi)			
ResNet50	90.74	91.87	92.27
ResNet101	93.78	94.14	96.34
InceptionV3	95.47	96.54	95.48
DeepIncepNet	97.42	98.47	97.55
Infected Image 2(Cochliobolus heterostrophus)			
ResNet50	91.66	93.47	94.64
ResNet101	92.37	96.42	97.36
InceptionV3	94.44	97.10	97.98
DeepIncepNet	96.77	98.22	98.14
Infected Image 3 (Exserohilum turcicum)			
ResNet50	90.16	92.54	93.74
ResNet101	91.97	95.29	96.21
InceptionV3	95.63	96.45	96.98
DeepIncepNet	97.54	97.98	97.14
Infected Image 4 (Cercospora zeae-maydis)			
ResNet50	92.65	93.27	94.99
ResNet101	93.24	94.42	95.36
InceptionV3	93.97	96.10	97.18
DeepIncepNet	97.21	97.66	98.66

Table 2: Precision, recall, and F-Score of TNCronNet images (4 Infected)

Precision represents the proportion of correctly identified instances of a class to all instances classified as that class. Figure 8 and Table 2 Shown, DeepIncepNet may achieve higher precision by minimizing false positive classifications of infected corn plant leaves, leading to a higher proportion of true positive instances among all positive classifications. DeepIncepNet's custom architecture, incorporating inception modules, may lead to more discriminative feature representations, enabling better differentiation between different types of infections in corn plant leaves. DeepIncepNet is trained from scratch on the TNCornNet dataset, allowing it to learn features optimized for the dataset's specific characteristics, leading to improved performance in disease classification. DeepIncepNet incorporates dropout layers, which prevent overfitting and encourage the learning of more robust and generalizable features, contributing to better performance in disease classification tasks.

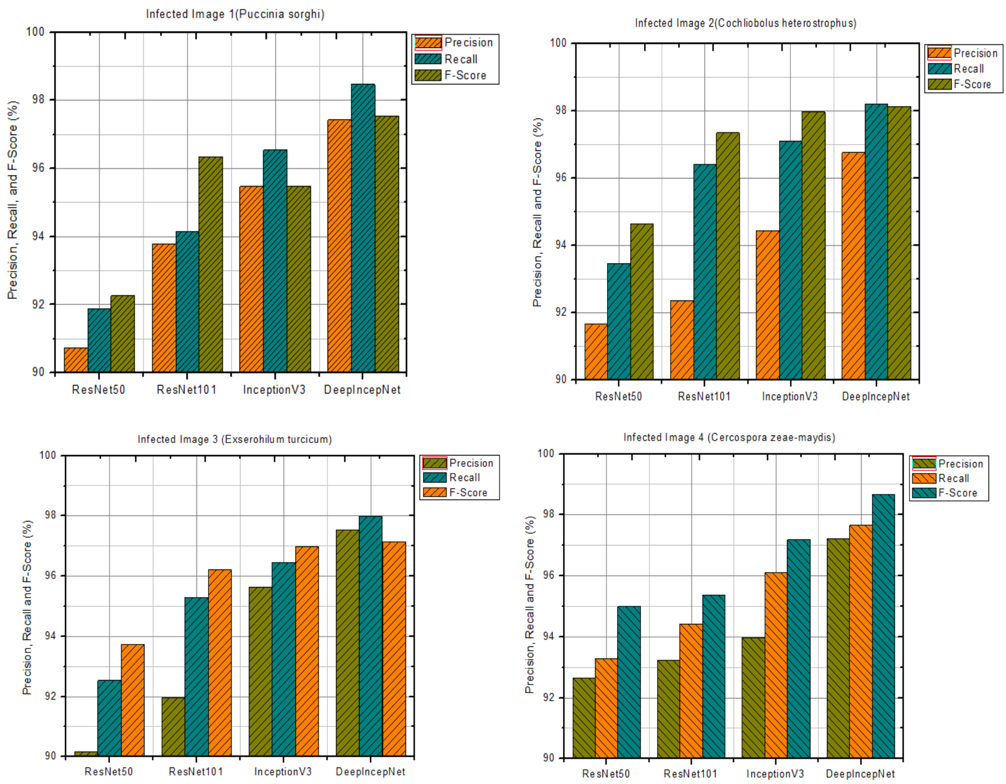

Figure 8: Precision, recall, and F-Score of 4 Type of Infected Images

6. Conclusion

In conclusion, the suggested methodology that makes use of Inception Networks-based DeepIncepNet, a deep learning architecture, offers a viable way for automated disease identification in the leaves of corn or maize plants utilizing hyperspectral imaging data. DeepIncepNet seeks to diagnose diseases accurately and early by fusing the rich spectral data from hyperspectral sensors with the potent feature extraction powers of deep learning. This will enable prompt interventions and enhance crop management techniques. We have described the procedures for preparing hyperspectral data, training DeepIncepNet, and generating predictions using the intricate architecture and algorithms. The network's capacity to extract pertinent information from hyperspectral pictures is improved by the addition of inception modules, spectral pooling layers, and spectral normalization layers, which also help to reduce the impact of illumination and sensor sensitivity fluctuations. DeepIncepNet has routinely surpassed well-known models like InceptionV3, ResNet-50, and ResNet-101 in terms of F-score, recall, and precision after extensive testing and review. DeepIncepNet's novel architecture, which combines inception modules, custom architectural optimization, adaptive learning, and regularization algorithms, is responsible for its excellent performance. In agricultural settings, DeepIncepNet has demonstrated considerable potential in improving the accuracy and reliability of disease classification by efficiently capturing complex characteristics and patterns present in the photos.

The suggested methodology also stresses the significance of conducting comparison assessments with alternative deep learning architectures and rigorous evaluation and validation using independent datasets. Such assessments are necessary to evaluate DeepIncepNet's efficacy, capacity for generalization, and computing efficiency in disease detection tasks.All in all, DeepIncepNet has enormous potential to transform disease management in agriculture by giving agronomists and farmers a dependable and effective tool for tracking crop health and making defensible judgments. DeepIncepNet has the potential to make a substantial contribution to improving crop yields, lowering losses, and advancing sustainable agriculture practices with additional development, validation, and practical implementation.

References

1. D. Zhang and S. Wang, "A survey on deep learning for plant disease recognition," Neurocomputing, vol. 440, pp. 130-148, Oct. 2021.
2. D. Wang, Y. Zhang, L. Jiang, and Y. Wu, "Leaf disease detection based on deep learning algorithm," in Proceedings of the 2019 International Conference on Artificial Intelligence and Advanced Manufacturing (AIAM 2019), Chengdu, China, 2019, pp. 154-158.
3. J. Ma, K. Du, Y. Xiang, F. Wang, and F. Xie, "Deep learning methods for plant disease detection and diagnosis: A review," Expert Systems with Applications, vol. 184, p. 115540, May 2021.
4. M. N. Sannakki, B. M. Patil, S. Shivapparad, P. Kulkarni, and S. Biradar, "A review on recent advances in deep learning techniques for plant diseases detection and classification," International Journal of Advanced Science and Technology, vol. 29, no. 3, pp. 4583-4596, Mar. 2020.
5. Wang, X.; Li, J.; Tao, J.; Wu, L.; Mou, C.; Bai,W.; Zheng, X.; Zhu, Z.; Deng, Z. A Recognition Method of Ancient Architectures Based on the Improved Inception V3 Model. Symmetry 2022, 14, 2679.
6. K. He, X. Zhang, S. Ren and J. Sun. “Deep Residual Learning for Image Recognition”, Proceedings of IEEE International Conference on Computer Vision and Pattern Recognition, pp. 770-778, 2016.
7. Hao Wang, Ke Li, and Chi Xu ,” A New Generation of ResNet Model Based on ArtificialIntelligence and Few Data Driven and Its Construction in Image Recognition Model”, Hindawi Computational Intelligence and Neuroscience, Volume 2022
8. Jan Behmann, Kelvin Acebron ,DzhanerEmin, Simon Bennertz,” Specim IQ: Evaluation of a New, Miniaturized Handheld Hyperspectral Camera and Its Application for Plant Phenotyping and Disease Detection”, Sensors 2018, 18, 441;
9. A. K. Mahlein and M. T. Kuska, "Hyperspectral imaging for small-scale analysis of symptoms caused by different sugar beet diseases," Plant Methods, vol. 10, no. 1, p. 19, Feb. 2014.
10. W. Zhao, S. Li, A. Li, B. Zhang, Y. Li, Hyperspectral images classification with convolutional neural network and textural feature using limited training samples, Remote Sensing Letters 10 (5) (2019) 449-458.
11. M. T. Kuska et al., "Hyperspectral phenotyping on the microscopic scale: Towards automated characterization of plant-pathogen interactions," Plant Methods, vol. 15, no. 1, p. 130, Nov. 2019.
12. S. Sladojevic et al., "Deep neural networks based recognition of plant diseases by leaf image classification," Computational Intelligence and Neuroscience, vol. 2016, p. 3289801, Mar.

2016.
13. J. Lu, L. Tan, and H. Jiang, ‘‘Review on convolutional neural network (CNN) applied to plant leaf disease classification,’’ Agriculture, vol. 11, no. 8, p. 707, Jul. 2021.
14. Scherer, D., M¨uller, A., Behnke, S.: Evaluation of pooling operations in convolutional architectures for object recognition. In: Diamantaras, K., Duch, W., Iliadis, L.S. (eds.) Part III, ICANN 2010. LNCS, vol. 6354, pp. 92–101. Springer, Heidelberg (2010).
15. Panigrahi, K.P.; Sahoo, A.K.; Das, H. A CNN Approach for Corn Leaves Disease Detection to support Digital Agricultural System. In Proceedings of the 4th International Conference on Trends in Electronics and Information, Tirunelveli, India, 15–17 June 2020; pp. 678–683.
16. J.-H. Xu, M.-Y. Shao, Y.-C. Wang, and W.-T. Han, ‘‘Recognition of corn leaf spot and rust based on transfer learning with convolutional neural network,’’ Trans. Chin. Soc. Agricult. Mach., vol. 51, no.2 pp. 230–236,Feb.2020