Deep Convolutional Neural Network (DCNN) Architectures and Transfer Learning for Robust Plant Seedlings Classification

Tayyab Bin Tahir
University of Oregon
Eugene, Oregon
ttahir@uoregon.edu

Abstract

The significance of agriculture for human existence and economic growth, particularly in underdeveloped and developing countries, cannot be overstated. With the growing world population and the challenges imposed by climate changes, there is a pressing need to increase food and cash crop production while minimizing costs. Earlier computer vision methods employed for selective weeding have faced significant difficulties in achieving reliable and accurate weed identification. In this paper, several approaches and architectures to classify plant seedlings are proposed using a dataset containing around 5,000 images of 960 unique plants belonging to 12 species at various stages of growth. The purpose of this study is to develop and compare several classification techniques to predict the plant species. The classification methods include Custom LeNet with different optimizations, and Transfer Learning with VGG, Residual Networks (ResNet), Inception_V3, DenseNet, SqueezeNet and EfficientNet models. By comparing the results and performances of these models we achieved ***95.26%*** *validation accuracy and aimed to identify the best model suitable for this dataset. The study also focuses on the data processing, parameter tuning, and layer construction of CNN models to gain insights into the computer vision field. The code is available at: https://github.com/RanaTayyab/Plant-Seedlings/*

1. Introduction

The utilization of precision agriculture has the capacity to bring about a significant transformation in contemporary farming practices through the customization of treatments in accordance with the specific variations and characteristics of the field. By way of illustration, the process of mapping local properties such as the ratio of crops to weeds can facilitate the early eradication of weeds, thus yielding advantageous outcomes from both an economic and environmental standpoint. The acquisition of data and the implementation of dependable analyses are pivotal in accurately gauging and quantifying local properties throughout the farmland.

The use of computer aided techniques for plant classification has gained significant importance due to the requirement for high classification accuracy and the need to save human resources. Research into plant classification utilizing computer vision methods dates back to 1995, where Brendel and Schwanke[1] implemented a knowledge-based object recognition approach for leaf classification. While transforming manually coded botanist features produced satisfactory results, the approach was limited in the number of classes it could handle, greatly affecting its practical application. Additionally, the need for manual feature conversion hindered the algorithm's expandability. To mitigate human intervention in feature extraction and automate the entire process, dimension reduction methods were proposed, significantly enhancing the feature selection procedure and expandability. However, these methods were still limited in terms of versatility and robustness. For instance, Golzaria and Frick[2] conducted research on differentiating three grass species using the PCA method but failed to achieve acceptable results in larger species sets. The advent of machine learning provides opportunities to improve image classification performance, including the classification of plant seedlings.

The conventional image classification approach involves the extraction of handcrafted features followed by a feature selection process, and the selection of an appropriate classifier. In contrast, convolutional neural networks (CNN) can learn a wide range of features from images, including both global and local features, and utilize them for effective classification. CNN has demonstrated superior performance compared to other image processing techniques. Hence, this paper focuses on the implementation of various Deep CNN architectures for plant seedling classification. The proposed system comprises four phases: preprocessing, construction of the network model architecture, training the network model, hyperparameter tuning, classification of plant seedlings and comparing the performance of various architectures. In the preprocessing phase, the input images undergo a series of preprocessing steps to enhance their quality and remove any potential noise. Next, the network model architecture

is constructed, followed by the training phase, where the CNN model is trained on a set of labeled data. Finally, the trained CNN model is utilized for the classification of plant seedlings.

The classification of seedlings presents a challenging task due to the complexity of the acquired scenes, which includes variations in illumination, similarities between weeds and crops at premature stages of growth, and intricacies in soil texture. Despite these complexities, CNN has demonstrated high classification performance. Therefore, this paper focuses on the development of a framework for a crop-weed discrimination system that employs CNN for the classification of 12 different plant species. Additionally, we compare the proposed seedling classification system by reproducing state-of-the-art techniques and architectures which include Custom LeNet with different image normalizations, transformations, optimizations, and Transfer Learning with VGG, Residual Networks (ResNet) [12], Inception_V3, DenseNet, SqueezeNet and EfficientNet models to determine the efficiency and effectiveness of the approach. The primary objective of this research is to enhance the accuracy and robustness of seedling classification systems, particularly in the context of crop-weed discrimination.

2. Related Work

In the field of plant classification, several studies have been conducted to develop effective models that can accurately distinguish between various species. Two strategies have been employed in the literature for the classification of crop and weed species. The first strategy involves segmenting images into green and soil regions, extracting features from green patches, and using classification techniques to obtain the specified classes. The second strategy involves utilizing deep learning techniques for plant seedling classification. Several works have been reported in this domain. For instance, [11] presented a method for classifying plant seedlings by consolidating the classification of the whole plants and the individual leaves. In this method, leaves are first separated from the plants, and features are extracted from both the whole plants and the segmented leaves. The classification process is performed for the leaves and plants, and finally, Bayes belief integration is used to fuse the classification results. Another work by Bakhshipour and Jafari [12] applied two significant pattern recognition approaches, artificial neural networks (ANN) and support vector machine (SVM), to separate the weeds from the sugar beet plants using shape features. Four species of prevalent weeds in the sugar beet fields were examined, and the results indicate that SVM slightly outperforms the ANN. In [13], the authors developed a system vision technique based on video processing and a hybrid ANN and ant colony algorithm classifier for assorting potato plant and three weed species. Texture features, obtained from the gray level co-occurrence matrix and the histogram, moment invariants, color features, and shape features are extracted. The Gamma test is then used to select the significant features.

Several studies have explored the application of deep learning techniques in plant classification, particularly in crop and weed identification. One study by [16] investigated the use of a convolutional neural network (CNN) to perform pixel-wise classification of maize, weeds, and soil. They modified the architecture of the VGG16 model, using a convolutional layer instead of a fully connected layer in the output. This approach resulted in the generation of semantic segmented images. Another study by Zhang et al. [9] explored the identification of broad-leaf weeds in pastures, comparing traditional machine learning techniques to deep learning approaches based on CNNs. They reported that the CNN approach achieved high accuracy and robustness in detecting weeds in real-world pasture environments. Additionally, [17] proposed a hybrid CNN model for classifying weed and crop species, combining elements of the AlexNet and VGGNET architectures. The model utilized normalization from AlexNet and filter depth selection from VGGNET. Furthermore, incremental learning was applied to enable the model to learn new plant species.

During the development of our model architectures, we examined previous works in the literature. One such work was conducted by [9], which focused on leaf classification using two popular convolutional neural network (CNN) architectures, namely GoogLeNet and VGGNet. In another study, [10] proposed a deep learning model with 26 layers and 8 residual blocks for the classification of 10,000 images of 100 ornamental plant species, achieving high classification rates of up to 91.78%.

3. Data Exploration

The dataset used in this paper was collected through a collaborative effort between the University of Southern Denmark and Aarhus University at the Aarhus University Flakkebjerg Research station [4]. It consists of annotated RGB images of approximately 960 unique plants belonging to 12 different species at various growth stages, with a physical resolution of approximately 10 pixels per millimeter [4]. The dataset is divided into a training set and a test set, with 80% of the images used for training and 20% used for validation, resulting in a total of 4750 images in the training set and 794 images in the test set. Researchers interested in accessing the dataset used in this study can download it from https://www.kaggle.com/c/plant-

seedlings-classification/data.

Figure 1. Total 12 Categorical Species

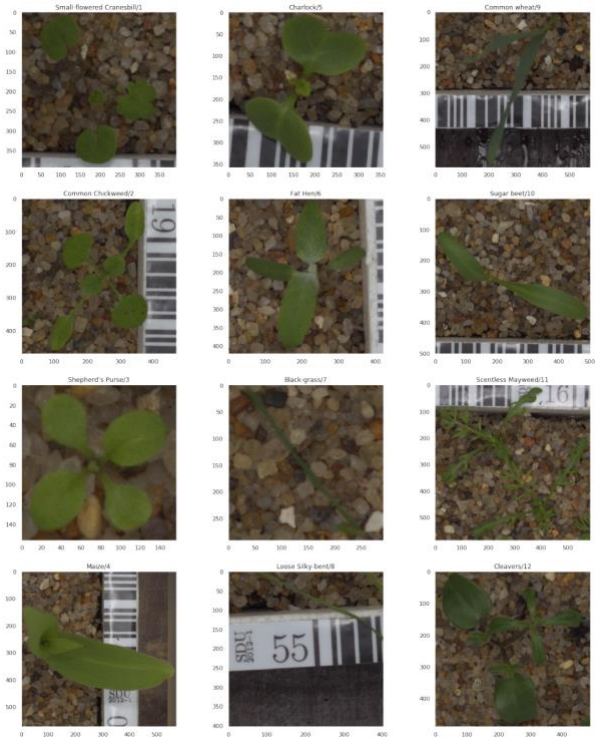

Figure 2. Zoomed Data Samples

Additionally, we describe the dataset's preprocessing techniques, which included data augmentation, picture segmentation, and label translation. Deep neural networks are used to autonomously extract features, doing away with the need for human feature extraction. The distribution of data among the various plant types in the collection is then examined, as seen in Figure 1. The findings show that the collection has some drawbacks, including a dearth of training examples for each species and an unbalanced sample size. In particular, "loose silky-bent" has 655 training samples, or 14% of the entire training set, compared to "common wheat," which has only 222 samples, or 5%.

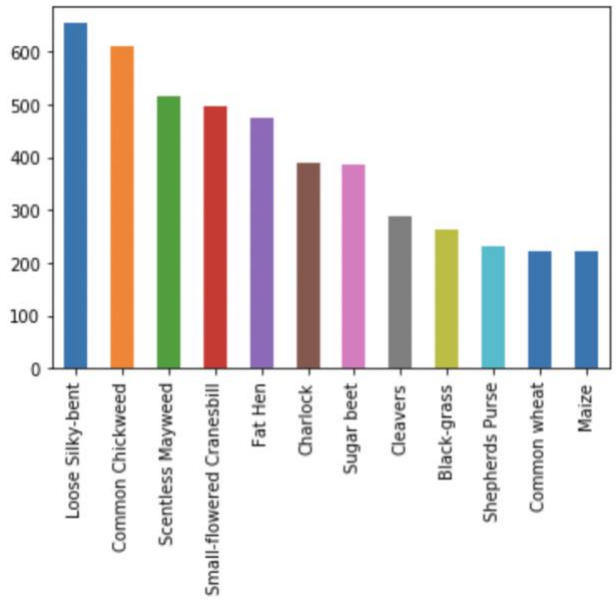

Figure 3. Data Distribution of Samples

4. Proposed DCNN Models

To find the optimal solution for this classification problem, multiple models were architected with several optimization functions, data transformations, data normalizations, batch sizes, hyperparameter tuning, number of layers and transfer learning techniques for fine-tuning the large architectures. Their performance was compared both in terms of training and validation accuracy to achieve the best possible results. The cross-entropy loss used for the models is represented as:

$$L(y,p) = -\frac{1}{N}\sum_{i=0}^{N-1}\sum_{j=0}^{K-1} y_{i,j} log(p_{i,j}) \quad (1)$$

4.1. LeNet with SGD

The LeNet architecture consists of several layers, including convolutional layers, pooling layers, and fully connected layers. The convolutional layers apply a set of filters to the input image, extracting features and reducing the dimensionality of the input. The pooling layers down

sample the output of the convolutional layers, further reducing the dimensionality of the input. The fully connected layers then perform the classification based on the extracted features.

Stochastic Gradient Descent (SGD) [12] is a widely used optimization algorithm for training neural networks, including LeNet. SGD is an iterative algorithm that updates the model parameters based on the gradients of the loss function with respect to the parameters, calculated on a small batch of training data at each iteration. This helps to reduce the computational cost and memory requirements of training, while still achieving good performance. In

LeNet with SGD, the weights of the network are updated using SGD during the training process. The loss function is typically chosen to be the cross-entropy loss, which measures the difference between the predicted class probabilities and the true class labels. The aim of training is to minimize the loss function by adjusting the weights of the network. The learning rate is an important hyperparameter in SGD, which determines the size of the steps taken in the weight updates. A too large learning rate can lead to oscillations or divergence, while a too small learning rate can result in slow convergence or getting stuck in a suboptimal solution.

```
class LeNet(nn.Module):
    def __init__(self, num_classes=12, num_rgb=3):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(num_rgb, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(44944, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, num_classes)

    def forward(self, x):
        out = F.relu(self.conv1(x))
        out = F.max_pool2d(out, 2)
        out = F.relu(self.conv2(out))
        out = F.max_pool2d(out, 2)
        out = out.view(out.size(0), -1)
        out = F.relu(self.fc1(out))
        out = F.relu(self.fc2(out))
        out = out.view(out.size(0), -1)
        out = self.fc3(out)
        return out
```

CNN Architecture #1. LeNet (To start with and enhanced later)

4.2. LeNet with SGD + Image Normalization

In LeNet with SGD and image normalization, the input images are first normalized using techniques such as zero mean normalization or standard deviation normalization, and then fed into the LeNet architecture for training. During training, the SGD optimizer updates the weights of the network based on the gradients of the loss function computed on a mini-batch of normalized images.

The combination of LeNet, SGD, and image normalization has been shown to be effective in achieving high accuracy in various classification tasks, especially in scenarios where the input images are subject to variations in lighting conditions, contrast, or color. Additionally, this combination can also help to reduce the training time and improve the convergence speed of the network, as the normalization step helps to reduce the scale of the input values and prevent the gradients from becoming too large or too small.

4.3. LeNet with SGD + Image Normalization + Data Augmentation

In LeNet with SGD, image normalization, and data augmentation, the input images are first normalized and then subjected to various data augmentations before being fed into the LeNet architecture for training. During training, the SGD optimizer updates the weights of the network based on the gradients of the loss function computed on a mini-batch of augmented images. By combining image normalization, data augmentation, and SGD, LeNet can achieve high accuracy in a variety of classification tasks, while also being robust to variations in the input data and resistant to overfitting. However, it's worth noting that selecting the appropriate augmentation techniques and parameters can require careful consideration and experimentation, as poorly chosen or excessive augmentations can lead to degraded performance or training instability.

4.4. LeNet with SGD + Image Normalization + Data Augmentation + Kernel Size 3

In convolutional neural networks, the kernel size refers to the size of the filters that are applied to the input images in the convolutional layers. A smaller kernel size, such as 3x3, allows the network to capture more detailed and local features in the input images, while also reducing the number of parameters and computational cost compared to larger kernel sizes.

In LeNet with SGD, image normalization, data augmentation, and kernel size 3, the convolutional layers use 3x3 filters to extract features from the input images. This can lead to better performance in some image classification tasks, particularly those that require the network to capture fine-grained features or distinguish between similar objects. Furthermore, the combination of image normalization, data augmentation, and SGD with kernel size 3 can help to improve the robustness and generalization performance of the network, while also

reducing overfitting and increasing training speed.

4.5. LeNet with SGD + Image Normalization + Data Augmentation + Kernel Size 3 + Batch Size 32

In addition to the techniques mentioned earlier, this variant of LeNet uses batch size 32 during training. The batch size refers to the number of samples processed in each iteration of the training process. A smaller batch size can lead to more frequent weight updates and better convergence, while a larger batch size can lead to improved computational efficiency and less noise in the weight updates.

In LeNet with SGD, transformation, image normalization, data augmentation, kernel size 3 [7], and batch size 32, the input images are first normalized and then subjected to various data augmentations and transformations before being fed into the LeNet architecture for training. During training, the SGD optimizer updates the weights of the network based on the gradients of the loss function computed on a mini-batch of 32 augmented images. The combination of transformation, image normalization, data augmentation, kernel size 3, and batch size 32 can help to improve the performance and robustness of the network by providing a diverse and representative set of training examples, while also ensuring that the network learns meaningful and generalizable features from the input images.

4.6. LeNet with Adam Optimizer + Image Normalization + Data Augmentation + Batch Size 32

The Adam optimizer is a popular optimization algorithm for deep learning that uses adaptive learning rates based on the gradient of the loss function. It has been shown to converge faster than traditional optimization methods, such as stochastic gradient descent (SGD), and can be more robust to noisy or sparse gradients.

In LeNet with Adam optimizer, transformation, image normalization, data augmentation, and batch size 32, the input images are first normalized and then subjected to various data augmentations and transformations before being fed into the LeNet architecture for training. During training, the Adam optimizer updates the weights of the network based on the gradients of the loss function computed on a mini-batch of 32 augmented images. The combination of Adam optimizer, transformation, image normalization, data augmentation, and batch size 32 can help to improve the performance and robustness of the network by providing a diverse and representative set of training examples, while also ensuring that the network learns meaningful and generalizable features from the input images. Moreover, the Adam optimizer can help to speed up the convergence of the network and reduce the need for manual tuning of the learning rate.

4.7. Transfer Learning with VGG16

Transfer learning is a technique in which a pre-trained model is used as a starting point for a new task, rather than training a model from scratch. In this variant of VGG16, the pre-trained model is initialized with weights from a model that was previously trained on a large dataset, such as ImageNet [8]. The last few layers of the model are then replaced or fine-tuned to adapt to the new task.

The input layer takes an RGB image of size 224 x 224 pixels as input. Convolutional layers: VGG16 consists of 13 convolutional layers, each followed by a rectified linear unit (ReLU) activation function and a 3 x 3 filter size. The first convolutional layer has 64 filters, and the subsequent layers have 128, 256, and 512 filters, respectively. There are two sets of 2 convolutional layers followed by a max-pooling layer with a 2 x 2 filter size. These max-pooling layers reduce the size of the feature maps and help to extract higher-level features. Fully connected layers: VGG16 has three fully connected layers of 4096 neurons each, followed by a ReLU activation function and a dropout layer with a dropout rate of 0.5. The last fully connected layer has 1000 neurons, corresponding to the 1000 classes in the ImageNet dataset. Softmax layer: The softmax layer is the output layer of the network, which produces a probability distribution over the 1000 classes in the ImageNet dataset.

In addition to transfer learning, this variant of VGG16 uses the SGD optimizer, transformation, image normalization, data augmentation, and batch size 32 during training. The input images are first normalized and then subjected to various data augmentations and transformations before being fed into the VGG16 architecture for training.

4.8. Transfer Learning with ResNet50

ResNet50 is a convolutional neural network architecture that consists of 50 layers and was proposed by Microsoft Research in 2015. It is known for its residual blocks, which allow for the learning of very deep architectures while avoiding the vanishing gradient problem. When combined with transfer learning, 30 epochs of training, SGD optimizer, transformation, image normalization, data augmentation, and batch size 32, ResNet50 can achieve high performance and accuracy in image classification tasks.

F(x):= H(x) - x which gives H(x) := F(x) + x (2)

The input layer takes an RGB image of size 224 x 224 pixels as input. Convolutional layers: The first layer is a 7 x 7 convolutional layer with 64 filters, followed by a max-pooling layer with a 3 x 3 filter size and stride 2. The next set of layers consists of 4 blocks, each containing multiple convolutional layers and a skip connection. Each block has a different number of filters, starting with 64 and doubling in size for each subsequent block.

Residual blocks: The key innovation in ResNet50 is the use of residual blocks, which enable the network to learn residual functions rather than trying to directly fit the underlying mapping. Each residual block contains two 3 x 3 convolutional layers, each followed by a batch normalization layer and a ReLU activation function. The input to the block is added to the output of the block to form the residual connection. The output of the block is then passed through another batch normalization and ReLU activation function before being passed on to the next layer.

Transfer learning involves using a pre-trained model as a starting point for a new task, rather than training a model from scratch. In this case, the pre-trained ResNet50 model is initialized with weights from a model that was previously trained on a large dataset, such as ImageNet. The last few layers of the model are then replaced or fine-tuned to adapt to the new task.

4.9. Transfer Learning with Inception_V3

The Inception_V3 architecture consists of multiple convolutional layers with different filter sizes, max pooling layers, and inception modules. The inception modules are composed of parallel convolutional layers with different filter sizes, which are concatenated together to form a single output. When performing transfer learning with Inception_V3, the architecture remains the same, but the final fully connected layer is replaced with a new output layer suitable for the new task. The pre-trained weights of the network are frozen up to a certain point, typically near the inception modules, and only the weights of the newly added layers are trained during the fine-tuning process.

Inception_V3 architecture: Input layer: accepts input images with size 299 x 299 x 3 Convolutional layers: a series of convolutional layers with different filter sizes, strides, and padding, followed by batch normalization and ReLU activation Inception modules: a series of parallel convolutional layers with different filter sizes, followed by concatenation and batch normalization Reduction layers: a combination of max pooling and convolutional layers used to reduce the spatial dimensions of the input Fully connected layers: a series of fully connected layers used to map the output of the convolutional layers to the desired output size Output layer: the final layer used for classification, regression, or other tasks depending on the specific application During transfer learning, the weights of the convolutional layers and inception modules are typically frozen, and only the weights of the fully connected layers and output layer are trained. By doing so, the pre-trained weights are used to extract relevant features from the input images, while the newly added layers are used to map these features to the output.

4.10. Transfer Learning with DenseNet121

DenseNet121 is a deep convolutional neural network architecture that is commonly used in computer vision tasks. It is composed of multiple dense blocks, each of which is composed of multiple convolutional layers that are densely connected to each other.

When performing transfer learning with DenseNet121, the architecture remains the same, but the final fully connected layer is replaced with a new output layer suitable for the new task. The pre-trained weights of the network are frozen up to a certain point, typically near the dense blocks, and only the weights of the newly added layers are trained during the fine-tuning process.

DenseNet121 architecture: Input layer: accepts input images with size 224 x 224 x 3 Convolutional layers: a series of convolutional layers with different filter sizes, strides, and padding, followed by batch normalization and ReLU activation. Dense blocks: a series of convolutional layers that are densely connected to each other, followed by batch normalization and ReLU activation Transition layers: a combination of convolutional and pooling layers used to reduce the spatial dimensions of the input and the number of channels Global average pooling layer: a pooling layer used to generate a fixed-size output regardless of the input size.

4.11. Transfer Learning with SqueezeNet

SqueezeNet is a lightweight convolutional neural network architecture that is designed to have a small memory footprint and low latency. When performing transfer learning with SqueezeNet, the architecture remains the same, but the final fully connected layer is replaced with a new output layer suitable for the new task. The pre-trained weights of the network are frozen up to a certain point, typically near the last convolutional layer, and only the weights of the newly added layers are trained during the fine-tuning process.

SqueezeNet architecture: Input layer: accepts input

images with size 227 x 227 x 3 Fire modules: a series of convolutional layers, called squeeze and expand layers, that are designed to reduce the number of parameters while maintaining high accuracy Pooling layers: a series of pooling layers used to reduce the spatial dimensions of the input and the number of channels. Convolutional layers: a series of convolutional layers used to capture the features of the input images. Global average pooling layer: a pooling layer used to generate a fixed-size output regardless of the input size. Fully connected layers: a series of fully connected layers used to map the output of the convolutional layers to the desired output size. Output layer: the final layer used for classification, regression, or other tasks depending on the specific application.

During transfer learning, the weights of the convolutional layers and pooling layers are typically frozen, and only the weights of the fully connected layers and output layer are trained. By doing so, the pre-trained weights are used to extract relevant features from the input images, while the newly added layers are used to map these features to the output.

4.12. Transfer Learning with EfficientNet-B1

EfficientNet-B1 is a convolutional neural network architecture that is designed to balance model size, accuracy, and computational efficiency. It is part of the EfficientNet family of models that was introduced in 2019 and has become popular due to its state-of-the-art performance on various computer vision tasks. When performing transfer learning with EfficientNet-B1, the architecture remains the same, but the final fully connected layer is replaced with a new output layer suitable for the new task. The pre-trained weights of the network are frozen up to a certain point, typically near the last convolutional layer, and only the weights of the newly added layers are trained during the fine-tuning process.

EfficientNet-B1 architecture: Input layer: accepts input images with size 240 x 240 x 3. Stem: a series of convolutional layers used to extract low-level features from the input images. Inverted bottleneck blocks: a series of building blocks that consist of a 1x1 convolutional layer, a depth wise separable convolutional layer, and a 1x1 convolutional layer, which are designed to increase the model's representational power while reducing the number of parameters. Head: a series of convolutional layers used to generate a fixed-size output regardless of the input size. Global average pooling layer: a pooling layer used to generate a fixed-size output regardless of the input size. Fully connected layers: a series of fully connected layers used to map the output of the convolutional layers to the desired output size. Output layer: the final layer used for classification, regression, or other tasks depending on the specific application.

During transfer learning, the weights of the convolutional layers and pooling layers are typically frozen, and only the weights of the fully connected layers and output layer are trained. By doing so, the pre-trained weights are used to extract relevant features from the input images, while the newly added layers are used to map these features to the output.

5. Experiments

5.1. Experimental System

The whole setup is being done on Google Colab with standard GPU. The machine learning and deep learning libraries involve PyTorch, and Scikit-Learn with various other data visualization techniques.

5.2. Data Preprocessing

Resizing Images. In order to prepare the dataset for training with a neural network, several preprocessing steps were taken. First, all images in the dataset were resized to a uniform size of 224x224 pixels.

Create Masks. Next, masks were created for each image to isolate the relevant portion of the plant from the background. These masks were generated using the HSV color-space, which allows for more accurate detection of color compared to the RGB color-space. The Hue channel in the HSV color-space is particularly useful for identifying different shades of a color. In contrast, the RGB color-space mixes the three primary colors of light to create different colors, making it less effective for color detection. The HSV color-space [6] is structured such that colors of each hue follow the radius of a circle, with the red primary at 0°, the green primary at 120°, and the blue primary at 240°. The saturation of each color can be reduced by tinting with white, which ranges from 1 to 0 on the horizontal axis. The value of each color, which describes the gray scale ranging from black to white, is mapped onto the vertical axis. The effectiveness of these preprocessing steps was evaluated to ensure that the resulting dataset was suitable for training with a neural network.

Morphological Techniques. Morphological operations are frequently employed in image processing, and one of the most used operations is closing. Closing is useful for filling small holes in images. An example of an image before and after applying closing.

Sharpen the Images. Sharpening an image enhances

the contrast between bright and dark regions, thereby highlighting features and making image texture more visible as shown in Figure 4.

Label Encode. Another common technique used in image classification is label encoding, which is performed using the LabelBinarizer. The label of each image is transformed into a binary vector representation of the corresponding class.

Train Test Split. To split the data into training and testing sets, we partition the data into 70% training and 30% testing. We also perform cross-validation to prevent overfitting. Half of the testing data is used for validation.

Image Pixel Normalization. Image normalization is a common preprocessing step in computer vision, especially for Convolutional Neural Networks (CNNs). It aims to standardize the pixel values of an image to make it easier for the CNN to learn from the data. Image normalization typically involves scaling the pixel values to be within a certain range, such as [0, 1] or [-1, 1], and subtracting the mean value of the pixel values from each pixel. This process helps to reduce the effect of illumination changes and variations in image contrast, which can make it difficult for the CNN to distinguish between different objects in the image. Additionally, normalization can help to speed up the training process and improve the accuracy of the model.

Data Augmentation. When the data distribution is not uniform or the number of samples is small, data augmentation can be used to generate additional images. Examples of data augmentation techniques include cropping, rotation, color manipulation [16], horizontal and vertical flipping, and resizing. The application of these techniques results in a larger training dataset. This technique helps to improve the performance of the model by providing more variety and diversity of images for the CNN to learn from. There are many libraries available for data augmentation, including OpenCV, and PyTorch.

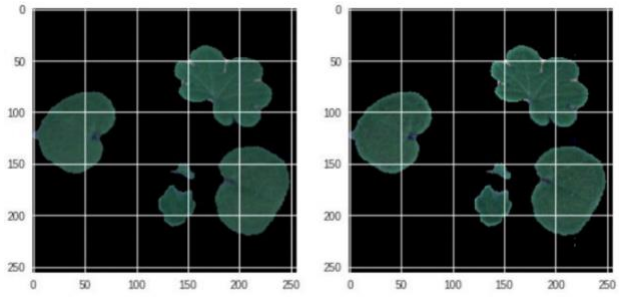

Figure 4. Image Sharpening

5.3. Experimental Model Comparisons

Evaluation Metric Accuracy is the ratio of total count of correctly classified plant seedlings (sum of TP (true positive) and TN (true negative)) over the total count of instances classified by the model (N designates the total instance count in the dataset).

$$\text{average accuracy} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{\text{total number of correct samples for i}^{\text{th}}\text{ class}}{\text{number of total samples of i}^{\text{th}}\text{class}}\right) \quad (3)$$

5.3.1. Lenet with SGD

Custom LeNet class trained on 70% of the training data and 30% on the test data. We achieved **12.74%** test accuracy and the training time took approximately 2 minutes and 44 seconds. The training loss, training accuracy, validation loss and validation accuracy are shown in the Figure 5.

5.3.2. Lenet with SGD + Image Normalization

Custom LeNet class with image data normalization trained on 70% of the training data and 30% on the test data. We achieved **61.68%** test accuracy and the training time took approximately 2 minutes and 41 seconds. The training loss, training accuracy, validation loss and validation accuracy are shown in the Figure 6.

5.3.3. Lenet with SGD + Image Normalization + Data Augmentation

Custom LeNet class with image data normalization and data augmentation trained on 70% of the training data and 30% on the test data. We achieved **62.32%** test accuracy and the training time took approximately 2 minutes and 53 seconds. The training loss, training accuracy, validation loss and validation accuracy are shown in the Figure 7.

5.3.4. Lenet with SGD + Image Normalization + Data Augmentation + Kernel Size 3

Custom LeNet class with image data normalization, data augmentation and a kernel size of 3 trained on 70% of the training data and 30% on the test data. We achieved **56.11%** test accuracy and the training time took approximately 2 minutes and 50 seconds. The training loss, training accuracy, validation loss and validation accuracy are shown in the Figure 8.

5.3.5. Lenet with SGD + Image Normalization + Data Augmentation + Kernel Size 3 + Batch Size 32

Custom LeNet class with image data normalization, data augmentation, a kernel size of 3 and a batch size of 32 trained on 70% of the training data and 30% on the test data. We achieved **86.95%** test accuracy and the training

time took approximately 1 hour, 32 minutes and 19 seconds for 100 epochs. The training loss, training accuracy, validation loss and validation accuracy are shown in the Figure 9(a).

5.3.6. Lenet with Adam + Image Normalization + Data Augmentation + Batch Size 32

Custom LeNet class using Adam optimizer with image data normalization, data augmentation, and a batch size of 32 trained on 70% of the training data and 30% on the test data. We achieved **87.47%** test accuracy and the training time took approximately 47 minutes and 25 seconds for 50 epochs. The training loss, training accuracy, validation loss and validation accuracy are shown in the Figure 9(b).

5.3.7. VGG16 Transfer Learning + 30 Epochs

VGG16 is fine-tuned using transfer learning technique which is trained on 70% of the training data and 30% on the test data. We achieved **95.26%** test accuracy and the training time took approximately 54 minutes and 40 seconds for 30 epochs. The training loss, training accuracy, validation loss and validation accuracy are shown in the Figure 10.

5.3.8. ResNet50 Transfer Learning + 30 Epochs

ResNet50 is fine-tuned using transfer learning technique which is trained on 70% of the training data and 30% on the test data. We achieved **93.89%** test accuracy and the training time took approximately 42 minutes and 13 seconds for 30 epochs. The training loss, training accuracy, validation loss and validation accuracy are shown in the Figure 11.

5.3.9. Inception_V3 Transfer Learning + 30 Epochs

Inception_V3 is fine-tuned using transfer learning technique which is trained on 70% of the training data and 30% on the test data. We achieved **94.95%** test accuracy and the training time took approximately 51 minutes and 14 seconds for 30 epochs. The training loss, training accuracy, validation loss and validation accuracy are shown in the Figure 12.

5.3.10. DenseNet_121 Transfer Learning + 30 Epochs

DenseNet_121 is fine-tuned using transfer learning technique which is trained on 70% of the training data and 30% on the test data. We achieved **95.26%** test accuracy and the training time took approximately 46 minutes and 26 seconds for 30 epochs. The training loss, training accuracy, validation loss and validation accuracy are shown in the Figure 13.

5.3.11. SqueezeNet Transfer Learning + 30 Epochs

SqueezeNet is fine-tuned using transfer learning technique which is trained on 70% of the training data and 30% on the test data. We achieved **94.11%** test accuracy and the training time took approximately 33 minutes and 51 seconds for 30 epochs. The training loss, training accuracy, validation loss and validation accuracy are shown in the Figure 14.

5.3.12. EfficientNet-B1 Transfer Learning + 30 Epochs

EfficientNet-B1 is fine-tuned using transfer learning technique which is trained on 70% of the training data and 30% on the test data. We achieved **91.26%** test accuracy and the training time took approximately 42 minutes and 29 seconds for 30 epochs. The training loss, training accuracy, validation loss and validation accuracy are shown in the Figure 15.

5.4. Results and Discussion

The results showed that with Transfer Learning VGG16 and DenseNet outperformed all the other models (achieving **95.26%** test accuracy). On the second place we can see Inception_V3 performed significantly. Another remarkable observation is that SqueezeNet achieves near to best accuracy (94.11%) but with significant less amount of time which is **33 minutes and 51 seconds**. See Figure 16.

Model	Train Accuracy	Test Accuracy	Time
LeNet with SGD	14.11%	12.74%	2 min, 44 sec
LeNet with SGD + Image Normalization	59.53%	61.68%	2 min, 41 sec
LeNet with SGD + Image Normalization + Data Aug	54.34%	62.32%	2 min, 53 sec
LeNet with SGD + Image Normalization + Data Aug + K=3	48.68%	56.11%	2 min, 50 sec
LeNet with SGD + Image Normalization + Data Aug + K=3 + B = 32	95.09%	86.95%	1 hr, 32 min, 19 sec
LeNet with Adam + Image Normalization	92.99%	87.47%	47 min, 25 sec

+ Data Aug + K=3 + B = 32			
VGG16 + Transfer Learning	98.28%	**95.26%**	54 min, 40 sec
ResNet50 + Transfer Learning	98.71%	93.89%	42 min, 13 sec
Inception_V3 + Transfer Learning	98.35%	**94.95%**	51 min, 14 sec
DenseNet121 + Transfer Learning	98.19%	**95.26%**	46 min, 26 sec
SqueezeNet + Transfer Learning	95.59%	**94.11%**	**33 min, 51 sec**
EfficientNet-B1 + Transfer Learning	97.17%	91.26%	42 min, 29 sec

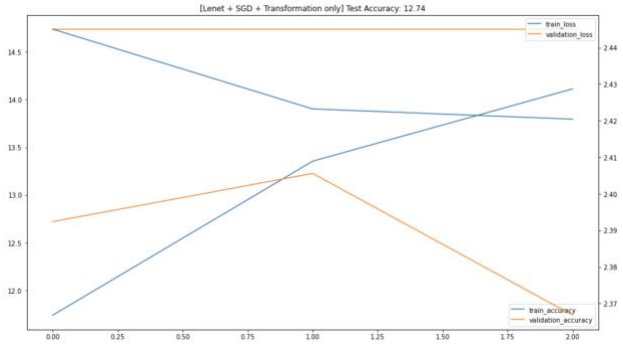

Figure 5. LeNet with SGD
Test Accuracy: 12.74%
Time: 2 min, 44 sec

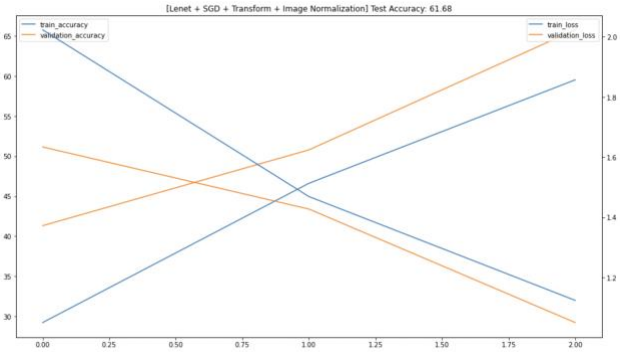

Figure 6. LeNet with SGD + Image Normalization
Test Accuracy: 61.68%
Time: 2 min, 41 sec

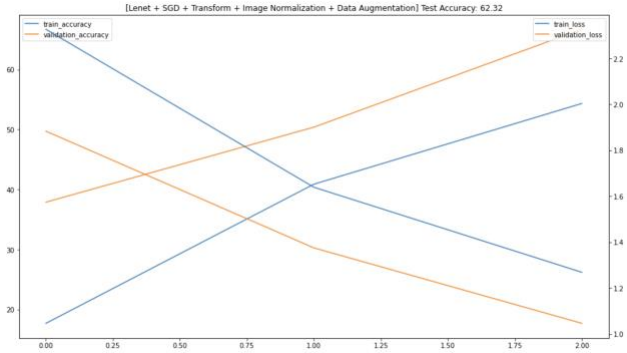

Figure 7. LeNet with SGD + Image Normalization + Data Augmentation
Test Accuracy: 62.32%
Time: 2 min, 53 sec

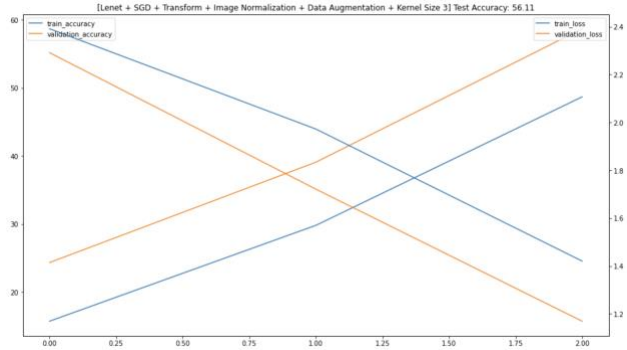

Figure 8. LeNet with SGD + Image Normalization + Data Augmentation + Kernel Size 3
Test Accuracy: 56.11%
Time: 2 min, 50 sec

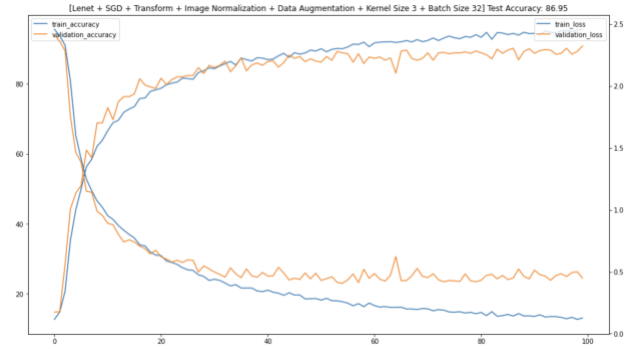

Figure 9(a). LeNet with SGD + Image Normalization + Data Augmentation + Kernel Size 3 + Batch Size 32 + 100 epochs
Test Accuracy: 86.95%
Time: 1 hr, 32 min, 19 sec

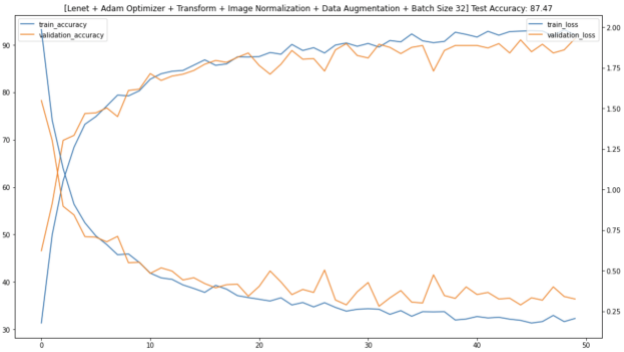

Figure 9(b). LeNet with Adam + Image Normalization + Data Augmentation + Batch Size 32 + 50 epochs
Test Accuracy: 87.47%
Time: 47 min, 25 sec

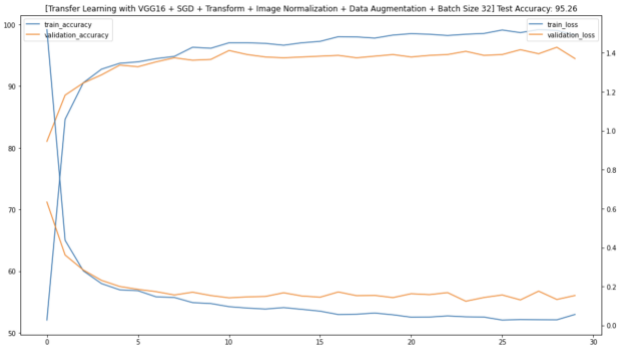

Figure 10. VGG16 with Transfer Learning + 30 epochs
Test Accuracy: 95.26%
Time: 54 min, 40 sec

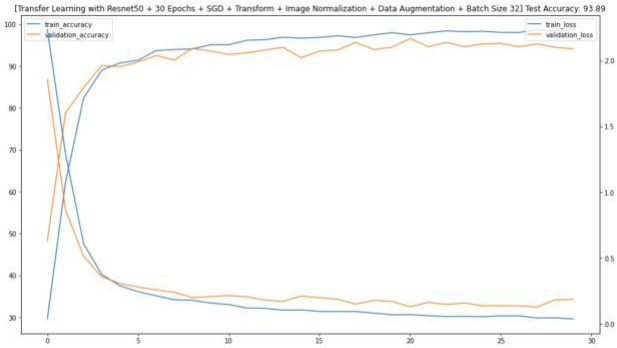

Figure 11. ResNet50 with Transfer Learning + 30 epochs
Test Accuracy: 93.89%
Time: 42 min, 13 sec

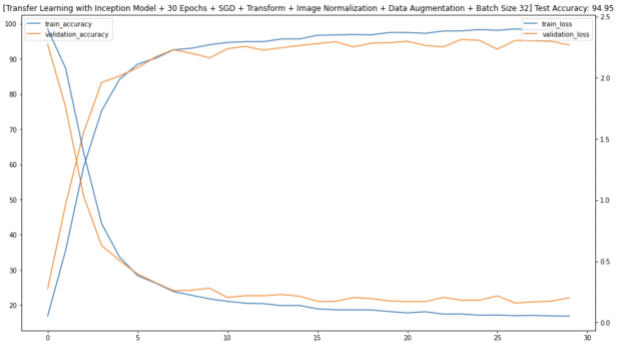

Figure 12. Inception_V3 with Transfer Learning + 30 epochs
Test Accuracy: 94.95%
Time: 51 min, 14 sec

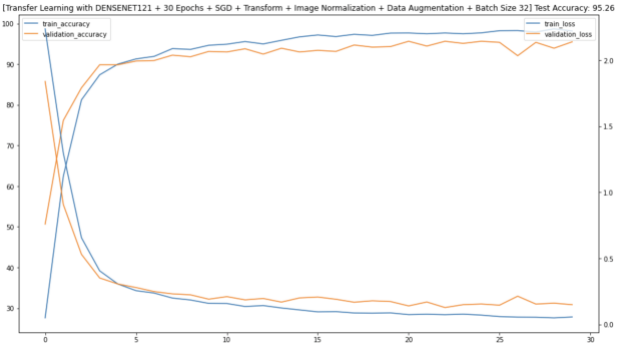

Figure 13. DenseNet121 with Transfer Learning + 30 epochs
Test Accuracy: 95.26%
Time: 46 min, 26 sec

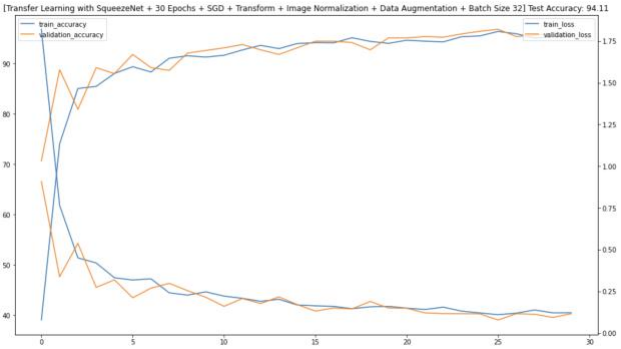

Figure 14. SqueezeNet with Transfer Learning + 30 epochs
Test Accuracy: 94.11%
Time: 33 min, 51 sec

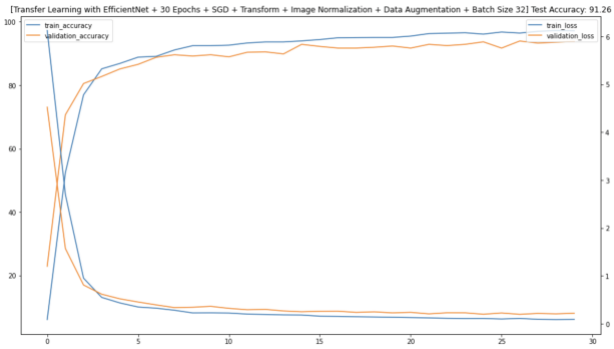

Figure 15. EfficientNet-B1 with Transfer Learning + 30 epochs
Test Accuracy: 91.26%
Time: 42 min, 29 sec

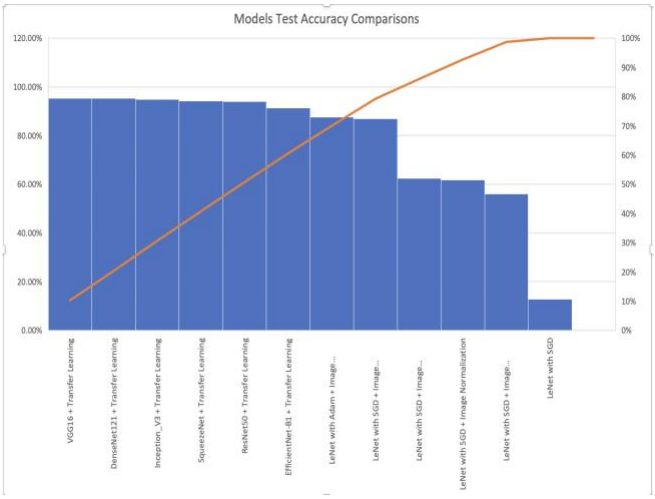

Figure 16. Models Test Accuracy Comparisons

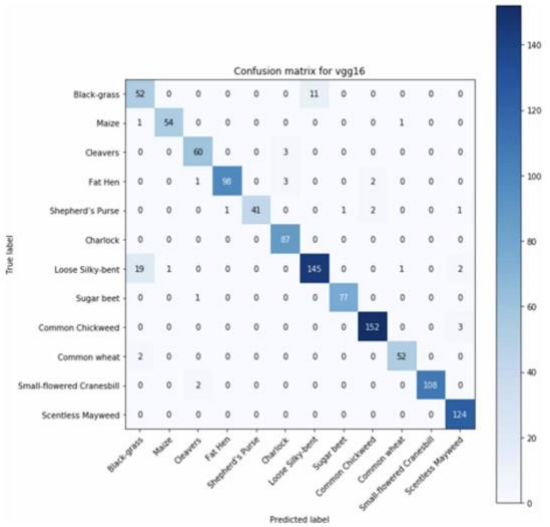

Figure 17. Confusion Matrix for VGG16

The results describe different experiments on the dataset using various deep learning models and techniques. The first four experiments use a custom LeNet class with different combinations of Stochastic Gradient Descent (SGD), Image Normalization, Data Augmentation, and Kernel Size 3, with Batch Size 32 used in the fifth experiment. The sixth experiment uses the Adam optimizer instead of SGD. The next five experiments involve Transfer Learning, with the VGG16, ResNet50, Inception_V3, DenseNet_121, and SqueezeNet models fine-tuned for 30 epochs. Finally, the EfficientNet model is fine-tuned for 30 epochs. Each experiment was trained on 70% of the training data and 30% of the test data, with accuracy and training time recorded. Figures are shown for each experiment depicting the training loss, training accuracy, validation loss, and validation accuracy. The best results were achieved using Transfer Learning with DenseNet_121, VGG16, and SqueezeNet models, which achieved test accuracies of 95.26%, 95.26%, and 94.11%, respectively with SqueezeNet being the best in training time model.

6. Conclusion

In conclusion, this paper presents an extensive evaluation of several deep convolutional neural network architectures and transfer learning techniques for robust plant seedlings classification. The proposed methods demonstrate promising results, achieving remarkable high accuracy rates in predicting the plant species. The comparative analysis of these models highlights the potential of transfer learning to improve classification performance while reducing the need for large amounts of data and computation resources. Furthermore, this research contributes to advancing the field of computer vision by providing insights into the data processing, parameter tuning, and layer construction of CNN models. The availability of the code on GitHub facilitates the reproducibility of the results and provides a valuable resource for researchers and practitioners. Overall, this study represents a significant step towards developing effective and efficient solutions for agriculture-related challenges, which can ultimately lead to sustainable and equitable economic growth.

References

[1] T. Brendel, J. Schwanke, P. F. Jensch, and R. Megnet, “Knowledgebased object recognition for different morphological classes of plants,” in Optics in Agriculture, Forestry, and Biological Processing, vol. 2345, pp. 277–284, International Society for Optics and Photonics,

1995.FirstName Alpher and FirstName Fotheringham-Smythe. Frobnication revisited. *Journal of Foo*, 13(1):234–778, 2003.

[2] T. M. Giselsson, H. S. Midtiby, and R. N. Jørgensen, "Seedling discrimination with shape features derived from a distance transform," Sensors, vol. 13, no. 5, pp. 5585–5602, 2013

[3] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 4700–4708, 2017.

[4] Philip J. Harrison1 Alexander Kensert1 and Ola Spjuth1. "Transfer Learning with Deep Convolutional Neural Networks for Classifying Cellular Morphological Changes". Original Research, 2019.

[5] W. Shang, S. Kihyuk, A. Diogo, and L. Honglak, "Understanding and improving convolutional neural networks via concatenated rectified linear units", International Conference on Machine Learning, pp. 2217-2225. 2016.

[6] C. Andrea, B. B. Mauricio Daniel and J. B. José Misael 2017 "Precise weed and maize classification through convolutional neuronal networks" IEEE 2nd Ecuador Technical Chapters Meeting (ETCM) pp 1-6

[7] McCool, C., Perez, T., & Upcroft, B. (2017). "Mixtures of Lightweight Deep Convolutional Neural Networks: Applied to Agricultural Robotics". IEEE Robotics and Automation Letters, 2(3), 1344–1351. doi:10.1109/ LRA.2017.2667039

[8] L.O.L.A. Silva, M.L. Koga, C.E. Cugnasca, and A.H.R. Costa, "Comparative assessment of feature selection and classification techniques for visual inspection of pot plant seedlings", Computers and Electronics in Agriculture vol. 97, pp. 47–55, 2013.

[9] Z. Qiu, J. Chen, Y. Zhao, S. Zhu, Y. He, and C. Zhang, "Variety identification of single rice seed using hyperspectral imaging combined with convolutional neural network", Applied Sciences, vol. 8, no. 2, 212, 2018.

[10] S. Ioffe, and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", The 32nd International Conference on Machine Learning (ICML 2015), Lille, France, July 6 –11, 2015.

[11] D. Nkemelu, D. Omeiza, and N. Lubalo, "Deep convolutional neural network for plant seedlings classification", Computer Vision and Pattern Recognition, arXiv: 1811.08404, 2018.

[12] X. Luo, D. Jayas, and S. Symons, "Comparison of statistical and neural network methods for classifying cereal grains using machine vision," Transactions of the ASAE, vol. 42, no. 2, p. 413, 1999.

[13] Azaab, S., Abu Naser, S., & Sulisel, O. (2000). "A proposed expert system for selecting exploratory factor analysis procedures". Journal of the College of Education 4 (2), 9-26.

[14] Aubert, B. A., Schroeder, A., and Grimaudo, J. 2012. "IT as Enabler of Sustainable Farming: An Empirical Analysis of Farmers' Adoption Decision of Precision Agriculture Technology," Decision Support Systems (54:1), pp. 510–520.

[15] Jeon, Wang-Su, and Sang-Yong Rhee, "Plant leaf recognition using a convolution neural network", International Journal of Fuzzy Logic and Intelligent Systems 17, no. 1, pp 26-34. 2017.

[16] A. Johannes, A. Picon, A. Alvarez-Gila, J. Echazarra, S. RodriguezVaamonde, A. D. Navajas, and A. Ortiz-Barredo, "Automatic plant disease diagnosis using mobile capture devices, applied on a wheat use case," Computers and electronics in agriculture, vol. 138, pp. 200–209, 2017.

[17] Olsen, A., Konovalov, D.A., Philippa, B.et al 2019 "DeepWeeds: A Multiclass Weed Species Image Dataset for Deep Learning" Sci Rep 9, 2058

[18] Khalid Ashraf, Bichen Wu, Forrest N. Iandola, Matthew W. Moskewicz, and Kurt Keutzer. "Shallow networks for high-accuracy road object-detection." arXiv:1606.01561, 2016.