Major in CS – DS
Major in CS – KP
Major in CS – MS
Major in CS – KV

Plant Disease Recognition using Spiking Neural Network

Dwijesh Jigarbhai Shah
Kush Jayank Pandya
Manan Shukla
Kunjan Vaghela

5/3/24
525 Brain Inspired Computing
Prof. Konstantinos Michmizos

Abstract

Plant diseases result in substantial economic losses and threaten food security in agriculture annually. The key to mitigating these losses lies in precisely identifying and promptly diagnosing plant diseases. While deep neural networks have been widely utilized for plant disease identification, they still encounter challenges such as low accuracy and a high number of parameters. The dataset used in this research project is publicly available on Kaggle data repository. The dataset consists of 1532 images of the dimensions of 4000 x 2500 which are classified as rusty, powdery and healthy. This research project has the potential to improve the accessibility of plant disease recognition for the farmers.

1. Introduction

In recent years, the agricultural sector has faced escalating challenges posed by plant diseases, which not only incur significant economic losses but also jeopardize global food security. The timely and accurate identification of these diseases stands as a critical imperative in mitigating their detrimental impact. While conventional machine learning techniques, particularly deep neural networks, have been extensively explored for this purpose, they often encounter limitations such as high computational complexity and suboptimal performance. In response to these challenges, a burgeoning field of research has emerged, focusing on the utilization of spiking neural networks (SNNs) for plant disease recognition.

SNNs represent a promising avenue in the realm of artificial neural networks, drawing inspiration from the biological mechanisms underlying information processing in the brain. Unlike traditional neural networks, which operate based on continuous activations, SNNs leverage discrete, time-varying spikes to encode and process information, mirroring the behavior of neurons in biological systems. This unique architecture offers several advantages, including inherent event-driven computation, low power consumption, and potential for parallelism, making SNNs particularly well-suited for real-time applications and energy-constrained environments.

Despite the potential they hold, the utilization of SNNs in identifying plant diseases has not been extensively explored, presenting an exciting opportunity for research and innovation. Our objective is to capitalize on the capabilities of SNNs to address the limitations of current methods and improve the precision and efficiency of plant disease detection systems. This study aims to examine the feasibility and effectiveness of integrating SNNs for this purpose, investigating innovative designs, learning techniques, and data representations tailored specifically to the unique attributes of plant disease datasets. Through thorough experimentation and assessment, our goal is to establish SNN-driven solutions as a practical and beneficial resource in combating plant diseases, thereby contributing to the sustainability and resilience of agricultural ecosystems globally.

2. Background (or Theory)

Plant disease recognition[1] using Spiking Neural Networks (SNNs)[2] presents a promising avenue for enhancing accuracy and efficiency in agricultural monitoring. However, it's essential to review existing literature to understand the strengths and limitations of alternative methodologies.

Diverse approaches such as employing various Convolutional Neural Network (CNN) architectures like VGG16, ResNet-50, etc., offer a strong foundation for disease recognition. These models leverage deep learning techniques and pre-trained weights to achieve impressive results. One advantage is their ability to capture complex patterns and features within plant images, enabling accurate disease identification. However, these approaches may suffer from computational complexity and require substantial resources for training and deployment, which could be a disadvantage, especially in resource-constrained environments.

Transfer learning is another widely adopted strategy that leverages pre-trained models on large datasets and fine-tunes them for disease recognition tasks. This approach significantly reduces the need for large annotated datasets and accelerates model convergence. Moreover, transfer learning facilitates knowledge transfer across domains, enabling the adaptation of models trained

on one plant species to another. Nonetheless, the efficacy of transfer learning heavily relies on the similarity between the source and target domains, posing a limitation when dealing with highly diverse plant species or diseases with unique characteristics.

Innovative techniques like EfficientNet-B0, C-DenseNet with CBAM, and M-bCNN represent advancements aimed at enhancing accuracy in disease detection, particularly in specific contexts such as wheat rust and leaf diseases. These techniques often incorporate novel network architectures, attention mechanisms, or feature fusion strategies to improve model performance. Their advantage lies in their ability to achieve state-of-the-art results on benchmark datasets, surpassing traditional CNN architectures. However, the implementation and fine-tuning of these advanced techniques may require specialized expertise and computational resources, limiting their accessibility and scalability in practical applications.

Overall, while diverse approaches and innovative techniques offer promising solutions for plant disease recognition using SNNs, each comes with its set of advantages and disadvantages. Understanding these nuances is crucial for selecting the most suitable approach based on the specific requirements of the application, available resources, and target plant species.

3. Experimental/Modeling Design

Dataset Preprocessing

The first step in our experimental design involves preprocessing the dataset to prepare it for training and validation. We utilize the Plant Disease Recognition Dataset, obtained from a publicly available source, which consists of images categorized into three classes: rusty, powdery, and healthy - sample images can be found as attached below. Each image has dimensions of 4000 x 2500 pixels.

Figure 1: Healthy, Powdery and Rusty Plant Leaves sample

We employ the following preprocessing steps:

- Image Resizing: Due to computational constraints and to standardize input sizes, we resize each image to a uniform size of 1024 x 1024 pixels.
- Data Augmentation: While not explicitly mentioned in the provided code snippet, data augmentation techniques such as rotation, flipping, and random cropping could be applied to augment the dataset and improve model generalization. However since we did not get expected improvement from the augmentation, we dropped this in our final code.

- Normalization: We normalize the pixel values of the images to have a mean of 0.485 and a standard deviation of 0.229. Normalization helps in stabilizing the training process by bringing all features to a similar scale.

Model Architecture

Our proposed model architecture is based on a Spiking Neural Network (SNN), which leverages the event-driven computation paradigm inspired by biological neurons. The model architecture is defined as follows:

- Convolutional Layers: The SNN architecture includes multiple convolutional layers, such as Conv2d, followed by ReLU activation functions. These layers are responsible for extracting hierarchical features from the input images.
- Pooling Layers: Max-pooling layers are employed to downsample the feature maps, aiding in capturing the most salient features while reducing spatial dimensions.
- Fully Connected Layers: The SNN incorporates fully connected layers, responsible for integrating the extracted features and making predictions. The final layer consists of three output neurons corresponding to the three disease classes.
- Leaky Integrate-and-Fire Neurons: Neurons in the SNN are modeled using leaky integrate-and-fire (LIF) dynamics, which simulate the behavior of biological neurons. These neurons accumulate membrane potential over time and emit spikes when a certain threshold is reached.

Training Procedure

The training procedure involves optimizing the SNN model parameters to minimize a custom-defined loss function, which combines Mean Squared Error (MSE) loss with spike-based counting metrics. The key components of the training procedure are as follows:

- Loss Function: We define a custom loss function that incorporates both traditional MSE loss and spike-based counting metrics. This loss function encourages the model to produce spike patterns consistent with the ground truth labels while also penalizing deviations from expected spike counts.
- Optimizer: We utilize the Adam optimizer with a learning rate of 1e-6 to update the model parameters during training. Adam is well-suited for training neural networks and helps in efficiently navigating the high-dimensional parameter space.
- Training Loop: During each epoch of training, we iterate over mini-batches of training data. For each mini-batch, we perform a forward pass through the SNN, compute the loss, backpropagate gradients, and update the model weights using the optimizer.
- Validation: After each epoch, we evaluate the model performance on a separate validation set. We compute validation loss and accuracy metrics to monitor the model's generalization performance and prevent overfitting.

Models Tried

1. Baseline Establishment: Normal CNN

To establish a baseline, we utilized a conventional Convolutional Neural Network (CNN) architecture.

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 224, 224]	1,792
MaxPool2d-2	[-1, 64, 112, 112]	0
Conv2d-3	[-1, 64, 112, 112]	36,928
MaxPool2d-4	[-1, 64, 56, 56]	0
Conv2d-5	[-1, 64, 56, 56]	36,928
MaxPool2d-6	[-1, 64, 28, 28]	0
Conv2d-7	[-1, 64, 28, 28]	36,928
MaxPool2d-8	[-1, 64, 14, 14]	0
Linear-9	[-1, 500]	6,272,500
Dropout-10	[-1, 500]	0
Linear-11	[-1, 500]	250,500
Dropout-12	[-1, 500]	0
Linear-13	[-1, 100]	50,100
Dropout-14	[-1, 100]	0
Linear-15	[-1, 3]	303

Total params	6,685,979
Trainable params	6,685,979
Non-trainable params	0

Table 1: Detailed Architecture Overview

2. Failed Approach: Norse Library

   Initially, we attempted to employ the Norse library [9] to simulate the Spike Neural Network (SNN) for the entire project. However, we encountered a significant issue

wherein the encoder failed to properly encode the image, leading us to explore alternative approaches.

3. Successful Approach: snntorch Library

   Details of the successful approach utilizing the snntorch library [10] with two different strategies:

   a. SNN with ANN:

      We attempted multiple models with ANN, specifically with grayscale image and colored input images with pixel size '256 x 256', '512 x 512' and '1024 x 1024'. However, we were unable to go much further due to infrastructure limitations, i.e. GPU RAM did not allow us to use bigger input images for processing.

   b. SNN with CNN and ANN:

      For this model as well we experimented with both grayscale and colored input images with different pixel sizes.

Layer (type)	**Output Shape**	**Param #**
Conv2d-1	[-1, 64, 254, 254]	23,296
MaxPool2d-2	[-1, 64, 126, 126]	0
Conv2d-3	[-1, 192, 126, 126]	307,392
MaxPool2d-4	[-1, 192, 62, 62]	0
Conv2d-5	[-1, 384, 62, 62]	663,936
MaxPool2d-6	[-1, 384, 30, 30]	0
Conv2d-7	[-1, 256, 30, 30]	884,992
MaxPool2d-8	[-1, 256, 14, 14]	0
Conv2d-9	[-1, 256, 14, 14]	590,080
MaxPool2d-10	[-1, 256, 6, 6]	0
Linear-11	[-1, 400]	5,018,000
RLeaky-12	[-1, 400]	0
Linear-13	[-1, 3]	1,203

RLeaky-14	[-1, 3]	0

Total params	7,488,899
Trainable params	7,488,899
Non-trainable params	0

Table 2: Detailed Architecture CNN proceeded by SNN with ANN

4. Results and Discussion

Overview of Experimental/Modeling Goals

Revisiting our experimental and modeling goals, our primary objective was to develop an effective model for plant disease classification using Spiking Neural Network (SNN) architectures. Inspired by biological neurons, our model incorporated convolutional layers, pooling layers, and fully connected layers, culminating in the implementation of leaky integrate-and-fire neurons. The training procedure involved optimizing model parameters using a custom-defined loss function and the Adam optimizer. With this framework in mind, we proceed to present and analyze our findings.

Data Analysis and Interpretation

The evaluation of our SNN model yielded mixed results, with notable challenges and areas for improvement:

Classification Accuracy:

- Baseline Establishment: The Normal Convolutional Neural Network (CNN) achieved a classification accuracy of 96%, serving as a strong baseline for comparison.
- Failed Approach: Utilizing the Norse Library resulted in an untrainable neural network due to issues with the encoder, resulting in equal outputs for all three classes.
- Successful Approach:
    - SNN with ANN: Validation accuracies for models ranged between 35%-45%, indicating moderate performance. However, this fell short of achieving comparable accuracy to the CNN baseline.
    - SNN with CNN and ANN: One standout model utilized a 1024x1024 input pixel size with a timestamp of 20, achieving a validation accuracy of 56%. While this represents an improvement, it still lags behind the CNN baseline.

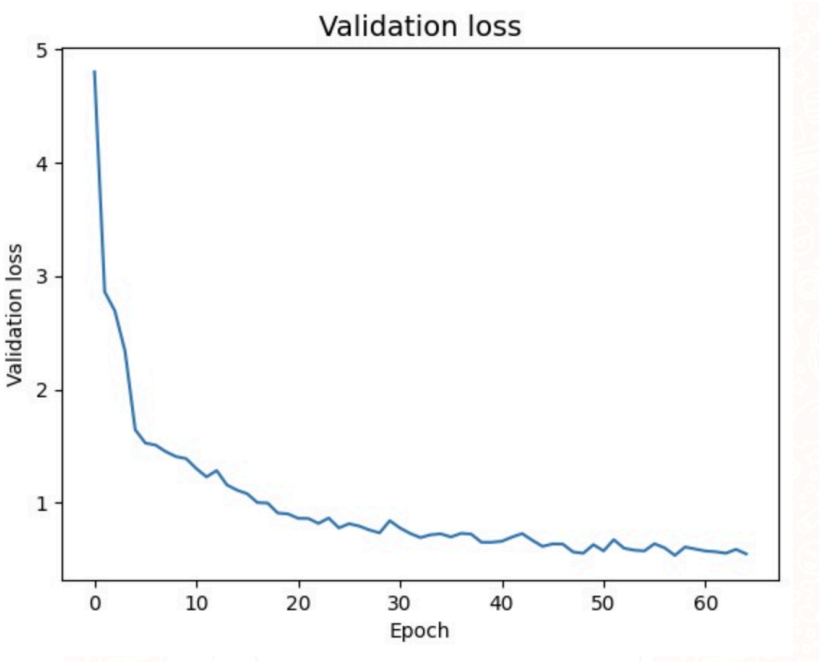

Figure 2: Validation Loss vs. Epoch Graph

Discussion of Findings:

- Room for Improvement: The less promising results of the SNN models highlight the need for further exploration and optimization. Achieving accuracy levels comparable to CNN demonstrates that there is still significant room for improvement in SNN architectures.
- Exploration of Architectures: Before considering SNN as the preferred model, additional exploration of architectures, hyperparameters, and training strategies is warranted. Fine-tuning the model parameters and investigating novel architectures may unlock the full potential of SNNs in plant disease classification.
- Future Directions: Future research should focus on addressing the limitations of SNN models, such as computational complexity and dataset biases. Incorporating advanced techniques, such as transfer learning and ensemble methods, could enhance the performance and robustness of SNN-based classifiers.

Describing Limitations and Uncertainty

Acknowledging the limitations and uncertainties, it's essential to consider inherent noise, variability, and computational constraints. While our SNN model showcases promising aspects, interpreting spike patterns remains subject to uncertainty. Computational constraints and dataset biases may also influence performance and generalization capabilities.

- Infrastructure Limitations: One significant limitation we encountered was related to infrastructure constraints. Specifically, our computational resources, particularly GPU RAM, imposed restrictions on the size of input images we could process. As a result, we were unable to explore larger input image sizes, which may have influenced the model's performance. This limitation constrained our ability to fully optimize the SNN architecture and may have impacted the achieved validation accuracy.
- Uncertainty in Performance: Despite our best efforts, our SNN models exhibited saturation in performance, with validation accuracy plateauing at around 45% to 50%.

This finding suggests that the model may have reached its performance limit within the constraints of our experimental setup. Further investigation is needed to determine whether alternative architectures or training strategies could overcome this performance bottleneck.

5. Conclusions

Our findings underscore the importance of exploring innovative approaches, such as Spiking Neural Networks (SNNs), in addressing the challenges of plant disease recognition. While our SNN models showed promise, their performance fell short of achieving accuracy levels comparable to conventional CNN architectures. This highlights the need for continued research and optimization efforts to fully harness the capabilities of SNNs in agricultural monitoring.

By shedding light on the potential and limitations of SNNs in plant disease recognition, our work underscores the significance of exploring alternative methodologies to enhance the precision and efficiency of disease detection systems. Furthermore, our study emphasizes the importance of understanding the nuances and complexities inherent in neural network architectures, paving the way for future advancements in this field.

Ultimately, our research contributes to the broader objective of improving food security and sustainability by facilitating early and accurate detection of plant diseases. By elucidating the opportunities and challenges associated with SNNs, we provide valuable insights that can inform the development of more robust and reliable disease classification models, ultimately benefiting farmers and agricultural stakeholders worldwide.

Acknowledgments

The author sincerely acknowledges and expresses gratitude towards the constant efforts and guidance provided by Prof. Konstantinos Michmizos. We also want to appreciate the help and support from Siwei Mai.

References

1. Liu, J., and Wang, X., 2021, "Plant diseases and pests detection based on deep learning: a review," Plant Methods, 17(1), p. 22. https://doi.org/10.1186/s13007-021-00722-9.
2. Zhang, H., Li, Y., He, B., Fan, X., Wang, Y., and Zhang, Y., 2023, "Direct training high-performance spiking neural networks for object recognition and detection," Frontiers in Neuroscience, 17. https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2023.1229951.
3. Krismer, J., Prausse, M., T., and Müller, T., 2016, "Epiphytic bacteria on native and cultivated plants in Raunkiær's climate zones of Europe," Frontiers in Plant Science, 7. https://doi.org/10.3389/fpls.2016.01419.
4. Tsalafoutas, I. A., and others, 2022, "Ionizing Radiation and Antioxidants: A Review of the Effects on Plants," Agriculture, 12(2). https://doi.org/10.3390/agriculture12020304.
5. Cao, X., and others, 2022, "Autism Spectrum Disorder from the Perspective of Brain Connectivity," Brain Sciences, 12(7). https://doi.org/10.3390/brainsci12070937.
6. Deng, L., and others, 2021, "Direct Learning-Based Deep Spiking Neural Networks: A Review," Article not yet indexed by journal details.

7. Wang, R., Zhang, W., Ding, J., Xia, M., Wang, M., Rao, Y., and Jiang, Z., 2021, "Deep Neural Network Compression for Plant Disease Recognition," Symmetry, 13(10). https://www.mdpi.com/2073-8994/13/10/1769.
8. Eshraghian, J. K., Ward, M., Neftci, E., Wang, X., Lenz, G., Dwivedi, G., Bennamoun, M., Jeong, D. S., and Lu, W. D., 2021, "Training Spiking Neural Networks Using Lessons From Deep Learning," CoRR, abs/2109.12894. https://arxiv.org/abs/2109.12894.
9. Pehle, C., & Pedersen, J. E. (2021). Norse - A deep learning library for spiking neural networks (Version 0.0.7) [Computer software]. Zenodo. https://doi.org/10.5281/zenodo.4422025
10. Eshraghian, J. K., Ward, M., Neftci, E., Wang, X., Lenz, G., Dwivedi, G., Bennamoun, M., Jeong, D. S., & Lu, W. D. (2023). Training spiking neural networks using lessons from deep learning. Proceedings of the IEEE, 111(9), 1016-1054.

Appendix

Code to create DataPipeline :

```
base_dir = '/kaggle/input/plant-disease-recognition-dataset'
train_dir = os.path.join(base_dir, 'Train', 'Train')
validation_dir = os.path.join(base_dir, 'Validation', 'Validation')

# Define transformations
if IS_GREYSCALE == True:
    transform = transforms.Compose([
        transforms.Grayscale(),  # Convert to grayscale
        transforms.Resize((IMAGE_SIZE, IMAGE_SIZE)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485], std=[0.229]),])
    channel = 1
else:
    transform = transforms.Compose([
        transforms.Resize((IMAGE_SIZE, IMAGE_SIZE)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485], std=[0.229]),])
    channel = 3

# Create datasets
train_dataset=datasets.ImageFolder(root=train_dir,transform=transform)
validation_dataset=datasets.ImageFolder(root=validation_dir,transform=tran
sform)
```

```
# Create data loaders
train_loader=DataLoader(train_dataset,batch_size=BATCH_SIZE,shuffle=True,
num_workers=4)
validation_loader=DataLoader(validation_dataset,batch_size=BATCH_SIZE,
shuffle=False, num_workers=4)
```

LOSS Function

```
loss_function = SF.mse_count_loss(correct_rate=0.8, incorrect_rate=0.2)
```

Neural Network

```
class Net(torch.nn.Module):
    """Simple spiking neural network in snntorch."""

    def __init__(self, timesteps, hidden, beta, img_size):
        super().__init__()

        self.timesteps = timesteps
        self.hidden = hidden
        self.beta = beta



        self.conv1 = nn.Conv2d(3, 64, kernel_size=11, stride=4, padding=2)
        self.conv2 = nn.Conv2d(64, 192, kernel_size=5, padding=2)
        self.conv3 = nn.Conv2d(192, 384, kernel_size=3, padding=1)
        self.conv4 = nn.Conv2d(384, 256, kernel_size=3, padding=1)
        self.conv5 = nn.Conv2d(256, 256, kernel_size=3, padding=1)

        self.pool = nn.MaxPool2d(kernel_size=3, stride=2)

        # layer 1
     self.fc1=torch.nn.Linear(in_features=12544,out_features=self.hidden)
     self.rlif1 = snn.RLeaky(beta=self.beta, linear_features=self.hidden)
```

```
        # layer 2

self.fc2=torch.nn.Linear(in_features=self.hidden,out_features=3)
  self.rlif2 = snn.RLeaky(beta=self.beta, linear_features=3)

    def forward(self, x):
        """Forward pass for several time steps."""

        # Initalize membrane potential
        spk1, mem1 = self.rlif1.init_rleaky()
        spk2, mem2 = self.rlif2.init_rleaky()

        # Empty lists to record outputs
        spk_recording = []

        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))
        x = self.pool(F.relu(self.conv4(x)))
        x = self.pool(F.relu(self.conv5(x)))
        x = torch.flatten(x, 1)



        for step in range(self.timesteps):
            spk1, mem1 = self.rlif1(self.fc1(x), spk1, mem1)
            spk2, mem2 = self.rlif2(self.fc2(spk1), spk2, mem2)
            spk_recording.append(spk2)

        return torch.stack(spk_recording)
```

Initialization

```
hidden = 400
```

```
device=torch.device("cuda")      if      torch.cuda.is_available()      else
torch.device("cpu")
model=Net(timesteps=20,hidden=hidden,beta=0.8,img_size=IMAGE_SIZE).to(devi
ce)

print(device)
```

Final Training loop

```
num_epochs = 65
optimizer = torch.optim.Adam(params=model.parameters(), lr=1e-6)
loss_hist = []
validation_accuracy_hist = []
validation_loss_hist = []



for epoch in range(num_epochs):
    model.train()
    minibatch_counter = 0
    epoch_loss = []

    # Training loop
    for     feature,     label     in     tqdm(train_loader,     desc=f'Epoch
{epoch+1}/{num_epochs}', leave=False):
        feature = feature.to(device)
        label = label.to(device)

        spk = model(feature) # forward-pass
        loss_val = loss_function(spk, label) # apply loss
        optimizer.zero_grad() # zero out gradients
        loss_val.backward() # calculate gradients
        optimizer.step() # update weights

        loss_hist.append(loss_val.item())
        epoch_loss.append(loss_val.item())
```

```
        minibatch_counter += 1

    avg_epoch_loss = sum(epoch_loss) / minibatch_counter

    # Compute validation accuracy and other metrics
    val_loss = []
    model.eval()
    with torch.no_grad():

        total = 0
        acc = 0
        for feature, label in validation_loader:
            feature = feature.to(device)
            label = label.to(device)

            spk = model(feature) # forward-pass
            loss = loss_function(spk, label)

            val_loss.append(loss)
            acc += SF.accuracy_rate(spk, label) * spk.size(1)
            total += spk.size(1)

    validation_loss = sum(val_loss)/len(val_loss)
    validation_loss_hist.append(validation_loss)

print(f"Epoch [{epoch+1}/{num_epochs}], Train Loss: {avg_epoch_loss:.4f},
Validation Loss: {validation_loss:.4f}, Validation Accuracy: {acc:.2f}%")
```