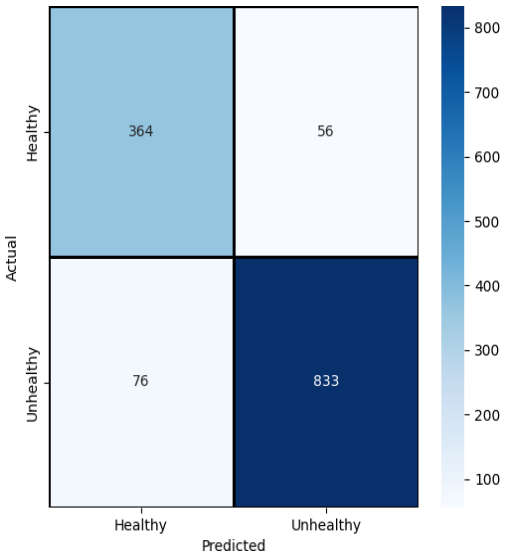
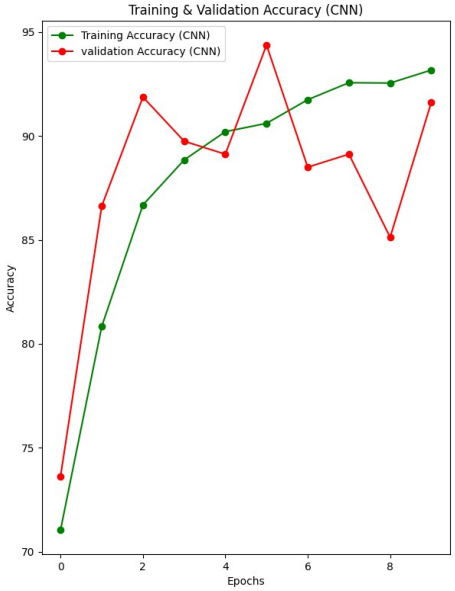
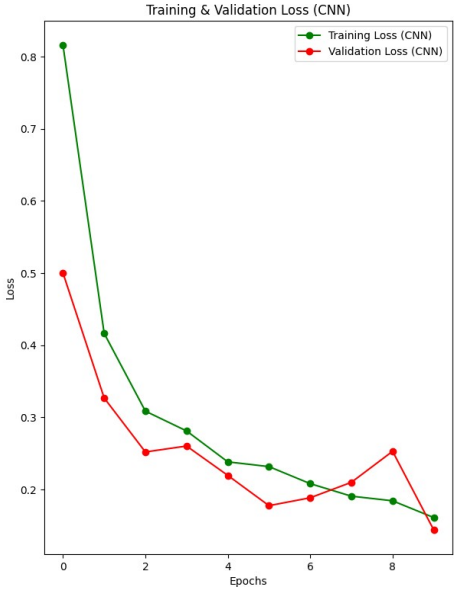
# CNN




First, we organized the dataset in a correct folder structure, so that we can use the ImageLoader. We categorized the data into 'healthy' and 'unhealthy' classes.

We trained our CNN model with the following hyperparameters: CNN(BATCH_SIZE=32, DROPOUT=0.5, LEARNING_RATE=0.001, NUM_EPOCHS=10, KERNEL_SIZE=3, PADDING=1). We used a CNN architecture with 3 Conv2d layers of sizes 3x16, 16x32, and 32x64, respectively. Each convolution layer was followed by a RELU activation layer, MaxPool2d layer (KERNEL_SIZE=2), and Dropout. At last, we had a fully connected layer of output size 256 followed by an output layer. We used the Adam optimizer and the CrossEntropy Loss function.

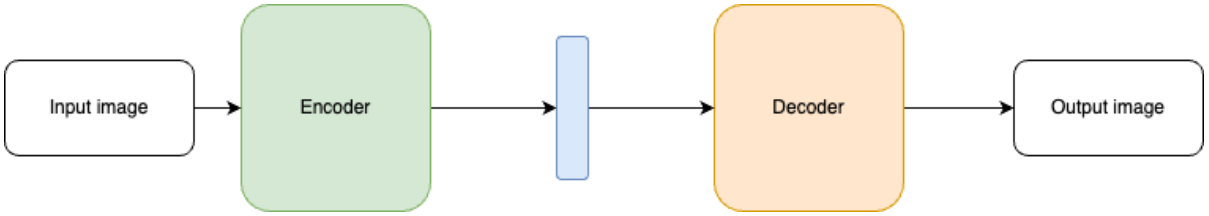



Looking at the curve, it appears that our CNN model has achieved a very high accuracy score and a minimal loss. The gap between the training and validation loss curves is not much, which suggests that our model did not overfit. But, these values can be misleading. We were working with a very small dataset. We wouldn't rule out overfitting so quickly. In fact, with the original dataset, our model did appear to overfit, with a training accuracy of 99% and a validation accuracy of 89%. Then, we augmented the training dataset by performing random transformations, such as shifting, rotation, noise addition, flipping, etc. Our model seemed to have performed well on the augmented dataset. The final accuracy we got was 93.4% on the training set and 90% on the validation set. The next logical step would be to reduce the complexity of our network and compare its performance to see if the current model is actually overfitted. Another thing we can do is get more data to train our model.

# Auto-Encoder

We've trained the autoencoder model on the healthy leaves dataset provided here. The dataset consists of both healthy and unhealthy images. We've only used healthy images for training the autoencoder. The architecture of the models is as shown below:




**Autoencoder architecture**

```
============== Encoder ==============
Sequential(
  (0): Conv2d(3, 12, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (1): ReLU()
  (2): Conv2d(12, 24, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (3): ReLU()
  (4): Conv2d(24, 48, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (5): ReLU()
)
============== Decoder ==============
Sequential(
  (1): ConvTranspose2d(48, 24, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (2): ReLU()
  (3): ConvTranspose2d(24, 12, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (4): ReLU()
  (5): ConvTranspose2d(12, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (6): Tanh()
)
```

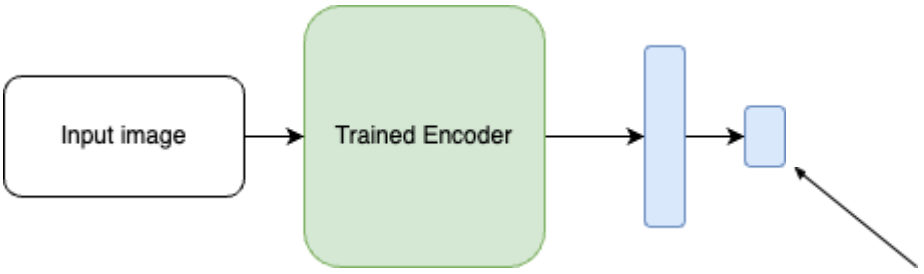
Hyperparameters

BATCH_SIZE=32, LEARNING_RATE=0.001, NUM_EPOCHS=10, KERNEL_SIZE=4, PADDING=1
Loss function = Mean squared error
Optimizer = Adam

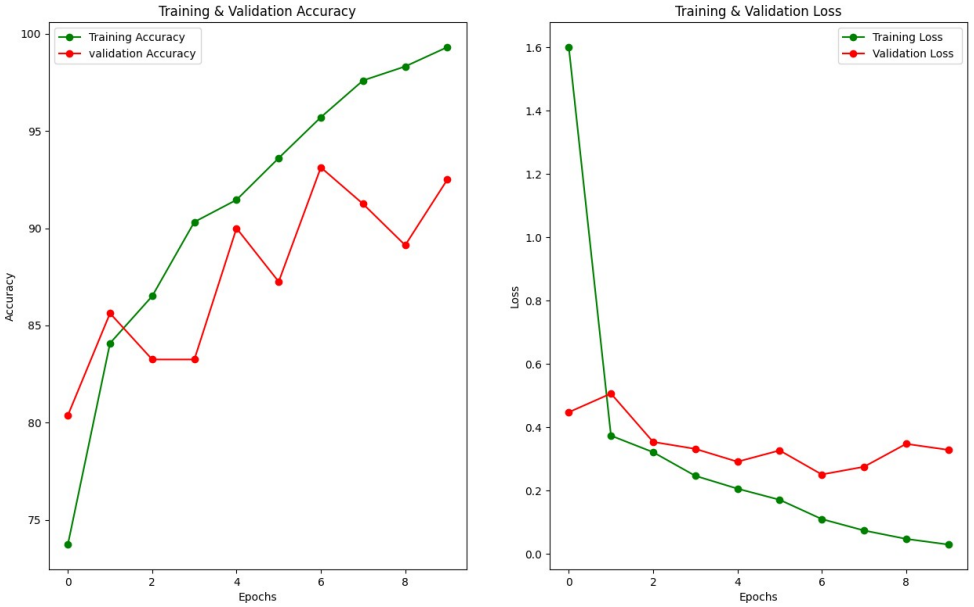
For classification, we're only using the encoder part of the trained autoencoder. We're using a fully connected layer of dimension 256 on top of the encoder. This layer is connected to another fully connected layer with 2 output nodes (number of classes). This model has then trained on the beans dataset for classifying leaf images as healthy or unhealthy.

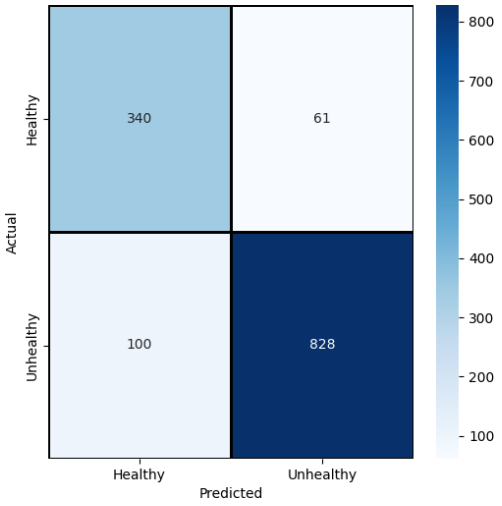
Hyperparameters

BATCH_SIZE=32, LEARNING_RATE=0.001, NUM_EPOCHS=10, KERNEL_SIZE=4, PADDING=1
Loss function = Cross entropy
Optimizer = Adam




**Classification layer**




We got an accuracy of 98.13 on the training set and 93 on the validation set. Overall the autoencoder-based model is performing better than CNN by 3 percent on the validation set. The reason for this could be that the encoder from the autoencoder knows how to extract features from a given leaf image. These features turn out to be helpful in the classification of healthy vs unhealthy leaves.

As we can see from the loss curves, there is a gap between the training and validation loss. This may be because the model is overfitting. We didn't use dropouts while training the autoencoder. Using dropout may help in fixing this issue.

Comparison between CNN and Autoencoder

| Model | Train accuracy | Valid accuracy |
|---|---|---|
| **Autoencoder** | **98.13** | **93** |
| CNN | 93.4 | 90 |

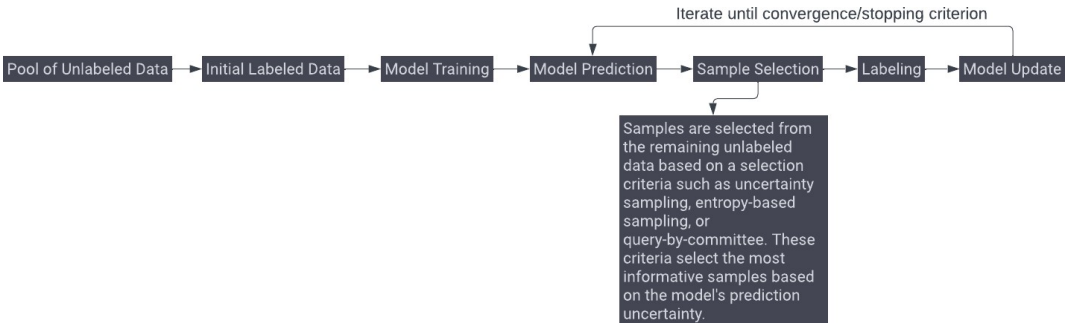
# Other Methods

***Different approaches that might effectively classify images with a small number of labeled***

***samples:***

1. **Transfer Learning:** Transfer learning involves using a pre-trained neural network on a large dataset and fine-tuning it on a small dataset for a specific task. This approach is effective because the pre-trained network has already learned features that are relevant for image classification, and fine-tuning helps to adapt those features to the specific task at hand. Sample architectures for this approach include VGG, ResNet, and Inception. Evaluation metrics can include accuracy, precision, recall, and F1-score.


2. **Semi-supervised Learning:** Semi-supervised learning involves using a small number of labeled samples along with a larger set of unlabeled samples to train a model. This approach is effective because it allows the model to leverage the information contained in the unlabeled samples to improve its performance. Sample architectures for this approach include variational autoencoders, generative adversarial networks, and ladder networks. Evaluation metrics can include accuracy, precision, recall, and F1-score.
3. **Domain Adaptation:** Domain adaptation involves adapting a model trained on a source domain to a new target domain with limited labeled data. This approach can be effective because it can transfer knowledge from the source domain to the target domain with minimal labeled data. One architecture for domain adaptation is the adversarial domain adaptation approach, which uses a discriminator to distinguish between source and target domains and a generator to generate domain-invariant features. This can be evaluated using metrics such as accuracy, precision, recall, and F1-score.
4. **Active Learning:** Active learning involves iteratively selecting the most informative samples from a large pool of unlabeled data and adding them to the labeled set to train a model. This approach can be effective because it can maximize the use of limited labeled data by selecting the most informative examples. One architecture for active learning is the uncertainty sampling approach, where the model selects the most uncertain examples for labeling. This can be evaluated using metrics such as accuracy, precision, recall, and F1-score.




5. **Few-Shot Learning:** Few-shot learning involves training a model on a small number of labeled samples and using that model to classify new samples with very few labeled examples. This approach is effective because it allows the model to learn quickly from a small amount of data. Sample architectures for this approach include Siamese networks, prototypical networks, and relation networks. Evaluation metrics can include accuracy, precision, recall, and F1-score.