1) Responses for the developed MLP model:
   A) Number of layers = 2 (1 hidden and 1 output)
   B) Total number of neurons = 74 (64 in hidden layer and 10 in output layer)
   C) Activation function used = Rectified Linear Unit (ReLu)

```python
# MLP architecture
class MLP(nn.Module):
    def __init__(self, num_input, hidden1_size, num_classes):
        super(MLP, self).__init__()
        self.hidden1 = nn.Linear(num_input, hidden1_size)
        self.output = nn.Linear(hidden1_size, num_classes)

    def forward(self, x):
        x = x.view(x.size(0), -1)
        h1_out = F.relu(self.hidden1(x))
        output = self.output(h1_out)
        return output
```

   D) Loss function used = Cross-entropy loss function is used

```python
criterion = nn.CrossEntropyLoss()
```

   E) Gradient method used: Stochastic Gradient Descent (SGD) is used as the gradient descent optimization algorithm.

   F) Hyperparameters used: Learning rate is set to 0.01, and momentum is set to 0.9

```python
mlp_optimizer = optim.SGD(mlp_model.parameters(), lr=0.01, momentum=0.9)
```

   G) Training method used: Batch-training method is used. Batch-size is set to 100.

```python
# MLP training
for epoch in range(5):
    mlp_train_loss, mlp_train_acc = train_model(mlp_model, train_loader, mlp_optimizer, criterion, device)
    print(f'MLP: Epoch [{epoch + 1}/5], Loss: {mlp_train_loss:.4f}, Accuracy: {mlp_train_acc:.2f}%')
```

```python
train_loader = DataLoader(dataset=train_dataset, batch_size=100, shuffle=True)
```

   H) Training time: Training time obtained for MLP mode is 87.53 seconds on CPU (AMD Ryzen 5 CPU)
   I) Percentage accuracy: An accuracy of 96.21% is obtained with test data.

```
MLP: Epoch [1/5], Loss: 0.4539, Accuracy: 86.78%
MLP: Epoch [2/5], Loss: 0.2388, Accuracy: 92.92%
MLP: Epoch [3/5], Loss: 0.1802, Accuracy: 94.68%
MLP: Epoch [4/5], Loss: 0.1459, Accuracy: 95.60%
MLP: Epoch [5/5], Loss: 0.1235, Accuracy: 96.40%
Training time for MLP model: 87.53 seconds
MLP: Test Loss: 0.1326, Test Accuracy: 96.21%
```

J) 64 neurons at the hidden layer strikes a good balance between complexity and performance. We use ReLU activation because it's known to work well in classification problems. Cross-entropy loss is suitable for tasks like digit recognition. We use SGD with momentum for its simplicity and effectiveness. Training with mini batches of size 100 speeds up training without sacrificing accuracy. We have chosen the learning rate and momentum through trial and error to get stable convergence and acceptable performance.

2) Responses for the developed CNN model:
   A) Number of convolutional layers: 2 convolutional layers.
   B) Kernel (filter) size: Kernal size for both the first and second convolutional layer is 5x5.
   C) Stride used: For both convolutional layer, 1 default stride is used.
   D) Number of filters: The first convolutional layer has 10 filters; the second convolutional layer has 20 filters.

```
self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
```

   E) Number of pooling layers: after every convolutional layer there is 1 pooling layer.
   F) Pooling size: For both pooling layers the pooling size is 2x2
   G) Pooling method used: Max pooling method.

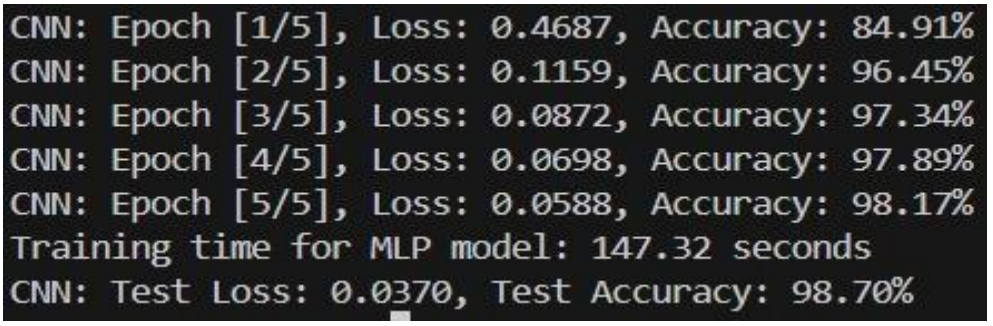
```
self.pool = nn.MaxPool2d(2)
```

   H) Number of layers and neurons for the fully connected layer: One complete connected layer with 50 neurons.

```
self.fc1 = nn.Linear(320, 50)
self.fc2 = nn.Linear(50, num_classes)
```

   I) Activation functions: The activation function that is used in the convolutional layers is ReLU, which is also complete connected layer.

   J) Loss functions used: Cross-entropy.
   K) Gradient methods used: Stochastic Gradient Descent with momentum.
   L) Hyperparameters: Momentum is set to 0.9 and learning rate is set to 0.01.
   M) Training methods used: Batch training that has minimum batch size of 100.
   N) Training time: Roughly 147.32 seconds on GPU is the training time for 5 epochs.
   O) Percentage accuracy: Accuracy is around 98.70%



P) Justification: A common CNN architecture for image classification uses two convolutional layers with ReLU activation, followed by max pooling to reduce dimensions while preserving key features. For multi-class classification, cross-entropy loss works well, and momentum-based SGD is selected because of its quick training. Hyperparameters are modified to create a compromise between convergence and precision. This approach usually produces outstanding performance and high accuracy on the MNIST dataset.

3) On the MNIST dataset, both the CNN and MLP models attain good accuracy. The CNN model, which included two convolutional layers followed by pooling layers, obtained even higher accuracy than the MLP model, which performed well with a reasonable training time and a single hidden layer of 64 neurons. ReLU activation, cross-entropy loss, and SGD with momentum were the only hidden layers that worked well for the MLP model. A 100-person mini-batch size batch training strategy was applied. Comparably, maintaining two convolutional layers with ReLU activation, max pooling, and suitable kernel sizes, in addition to cross-entropy loss and SGD with momentum, improved the performance of the CNN model.

   Adjusting hyperparameters such as learning rate and momentum and or using other Gradient Descent methods such as “Adam” may further improve the model's training performance albeit at the cost of accuracy.

4) Data Collection:
   We have captured the images by going into the CSUF Arboretum and labeled the dataset using the labeling application.

   Data Preprocessing:
   We have converted the images into 244x244 resolution and split the dataset into train and test directories.

   Model Selection:
   We have selected ResNet as a pre-trained CNN model because of its popularity and efficiency.

   Training Approach:
   We employed a transfer learning approach, utilizing the pre-trained ResNet model. The final layers of ResNet were retrained on the Arboretum dataset.

Conclusion: In conclusion, the ResNet model shows promise in classifying trees and plants from the CSUF Arboretum dataset. Further optimization and experimentation are recommended to enhance the model's performance.

Responses for the developed CNN model using dataset collected from Fullerton Arboretum:

A) Number of convolutional layers: ResNet-18 has 18 layers including convolutional layers
B) Kernel (filter) size: Default Kernel size of ResNet18 – 3*3

```
model = torchvision.models.resnet18(pretrained=True)
```

C) Stride used: Default stride 1 is used.
D) Number of filters: The first convolutional layer has 10 filters; the second convolutional layer has 20 filters.
E) Number of pooling layers: after every convolutional layer there is 1 pooling layer.
F) Pooling size: For both pooling layers the pooling size is 2x2
G) Pooling method used: Max pooling method.
H) Number of layers and neurons for the fully connected layer: One complete connected layer with 50 neurons.
I) Activation functions: The activation function that is used in the convolutional layers is ReLU. Softmax for output layer.
J) Loss functions used: Cross-entropy.
K) Gradient methods used: Stochastic Gradient Descent with momentum.
L) Hyperparameters: Momentum is set to 0.9 and learning rate is set to 0.001.
M) Training methods used: Batch training that has minimum batch size of 32.
N) Training time: Roughly 79.56 seconds.
O) Percentage accuracy: Accuracy is around 59.50%

```
Training time for CNN model: 79.56 seconds
Accuracy: 59.5%
```

P) Justification: ResNet18 is chosen for its effectiveness in image classification tasks. Hyperparameters were chosen based on trial and error and may require more tuning for optimal performance.