In [1]:

```
import torch
import torchvision
from torchvision import datasets,transforms
from torchvision.datasets import ImageFolder
from torch.utils.data import DataLoader,random_split
from PIL import Image
```

In [2]:

```
data = "C:/Users/sastra/Desktop/LAB/GrapeVine/Grapevine_Leaves_Image_Dataset"
```

In [3]:

```
transform = transforms.Compose([
    transforms.Resize((100, 100)),  # Resize images to 100x100
    transforms.ToTensor(),  # Convert images to PyTorch tensors
    transforms.RandomHorizontalFlip(),  # Randomly flip images horizontally
    transforms.RandomRotation(20),  # Randomly rotate images by 20 degrees
])
```

In [4]:

```
dataset = ImageFolder(root=data,transform = transform)
```

In [5]:

```
num_classes = len(dataset.classes)
num_classes
```

Out[5]:

```
5
```

In [6]:

```
train_size = int(0.7 * len(dataset))
val_size = int(0.2 * len(dataset))
test_size = len(dataset) - train_size - val_size
```

In [7]:

```
train_dataset, val_dataset, test_dataset = random_split(dataset, [train_size, val_size, test_size]
```

In [8]:

```
len(train_dataset),len(val_dataset),len(test_dataset)
```

Out[8]:

```
(350, 100, 50)
```

In [9]:

```
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

In [10]:

```
for images, labels in train_loader:
    print(images.shape, labels.shape)
    break
```

```
torch.Size([32, 3, 100, 100]) torch.Size([32])
```

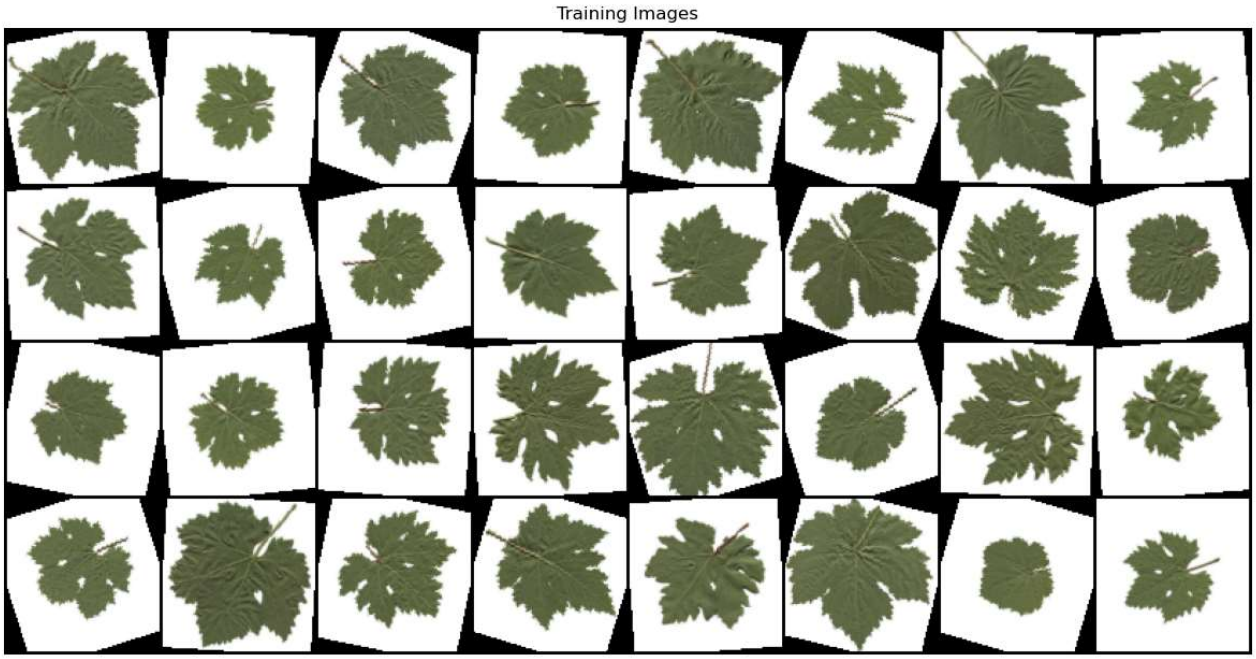
In [11]:

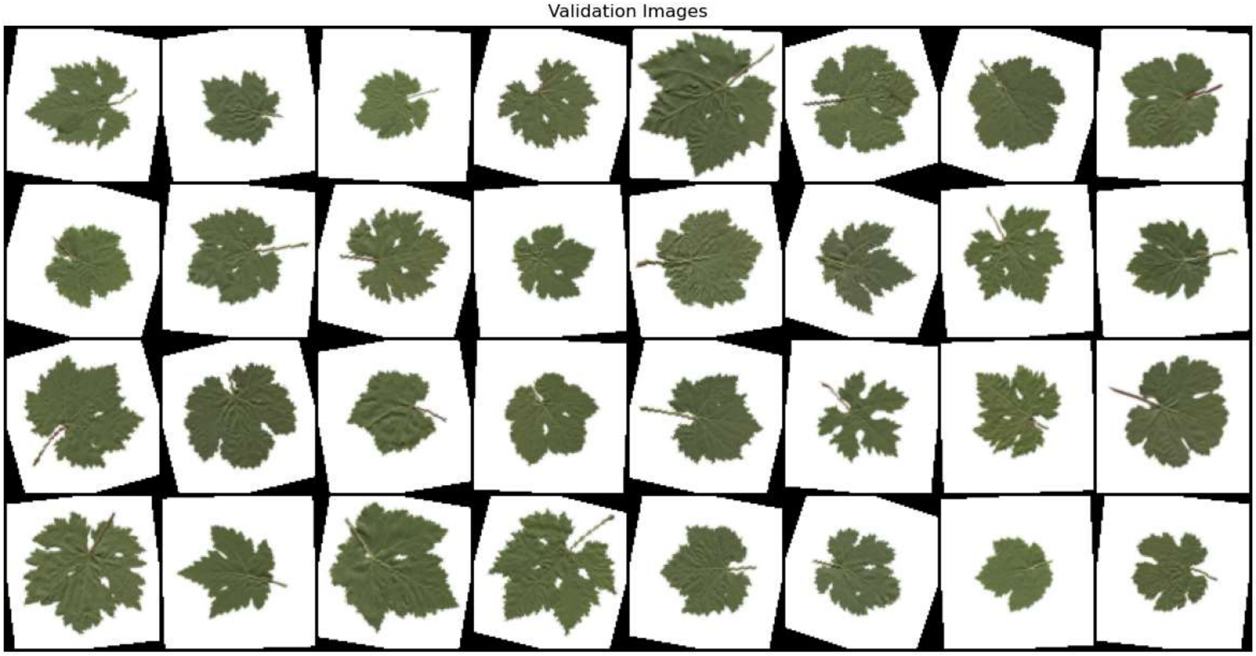
```
data_iter = iter(train_loader)
images, labels = next(data_iter)

print(f"Batch size: {images.shape}")
print(f"Labels: {labels}")
```

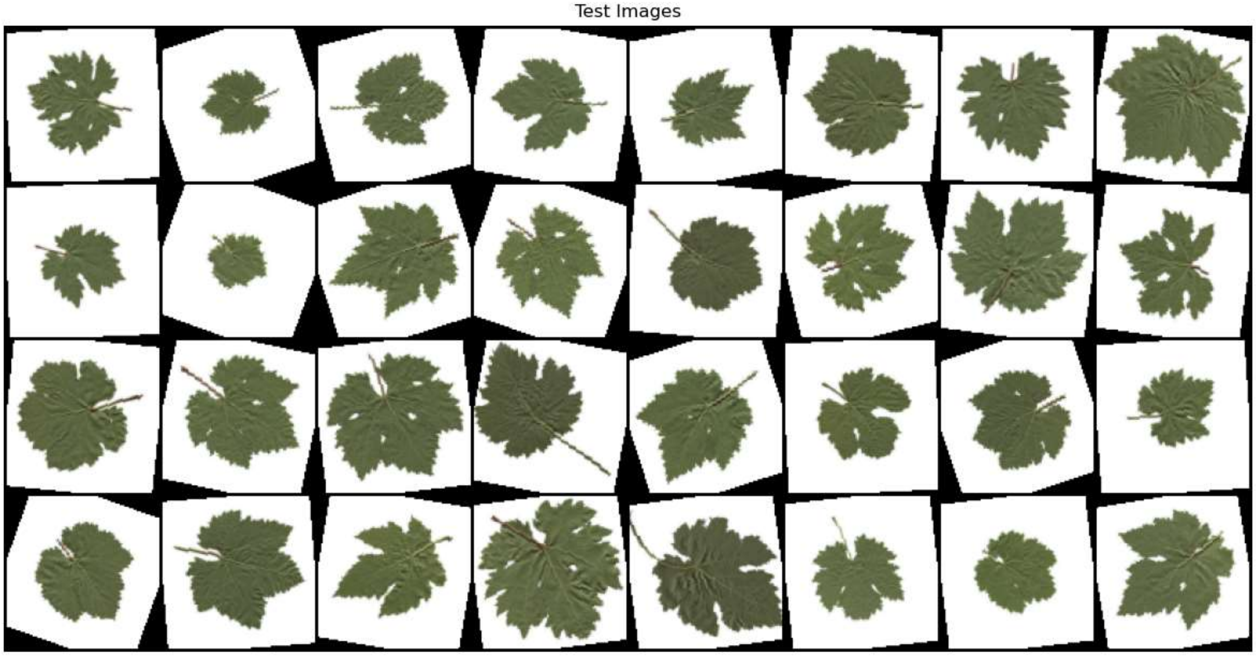
```
Batch size: torch.Size([32, 3, 100, 100])
Labels: tensor([4, 1, 3, 1, 4, 4, 2, 2, 2, 0, 3, 3, 2, 0, 2, 2, 3, 2, 1, 3, 1, 1, 4, 3,
        0, 2, 2, 3, 1, 2, 3, 2])
```

In [12]:

```
import matplotlib.pyplot as plt
import numpy as np
import torchvision

# Function to show a batch of images
def show_images(loader, title):
    # Get a batch of images
    images, labels = next(iter(loader))

    # Make a grid from batch
    grid = torchvision.utils.make_grid(images, nrow=8, padding=2)

    # Convert to numpy and transpose from (C, H, W) to (H, W, C) format
    grid = np.transpose(grid.numpy(), (1, 2, 0))

    # Plot the grid
    plt.figure(figsize=(15, 15))
    plt.imshow(grid)
    plt.title(title)
    plt.axis('off')
    plt.show()

# Show a batch of training images
show_images(train_loader, 'Training Images')

# Show a batch of validation images
show_images(val_loader, 'Validation Images')

# Show a batch of test images
show_images(test_loader, 'Test Images')
```







In [13]:
```
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
```

In [14]:
```
class CNN(nn.Module):
    def __init__(self,num_classes):
        super(CNN,self).__init__()
        self.conv1 = nn.Conv2d(3,32,kernel_size=3, padding=1)
        self.bn1 = nn.BatchNorm2d(32)
        self.pool = nn.MaxPool2d(2,2)

        self.conv2 = nn.Conv2d(32,64,kernel_size=3, padding=1)
        self.bn2 = nn.BatchNorm2d(64)

        self.dropout = nn.Dropout(0.5)

        self.fc1 = nn.Linear(64*25*25,124)
        self.fc2 = nn.Linear(124,num_classes)

    def forward(self,x):
        x = self.pool(F.relu(self.bn1(self.conv1(x))))
        x = self.pool(F.relu(self.bn2(self.conv2(x))))

        x = x.view(x.size(0), -1)
        x = self.dropout(x)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)

        return x
```

In [15]:

```python
def train_model(model, train_loader, val_loader, criterion, optimizer, n_epochs, device):
    train_losses, val_losses = [], []
    train_accs, val_accs = [], []

    model.to(device)

    for epoch in range(n_epochs):
        # Training phase
        model.train()
        total_train_loss, correct_train, total_train = 0, 0, 0

        for images, labels in train_loader:
            images, labels = images.to(device), labels.to(device)
            optimizer.zero_grad()

            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            total_train_loss += loss.item()
            _, preds = torch.max(outputs, 1)
            correct_train += (preds == labels).sum().item()
            total_train += labels.size(0)

        train_loss = total_train_loss / len(train_loader)
        train_acc = correct_train / total_train
        train_losses.append(train_loss)
        train_accs.append(train_acc)

        # Validation phase
        model.eval()
        total_val_loss, correct_val, total_val = 0, 0, 0

        with torch.no_grad():
            for images, labels in val_loader:
                images, labels = images.to(device), labels.to(device)
                outputs = model(images)
                loss = criterion(outputs, labels)
                total_val_loss += loss.item()

                _, preds = torch.max(outputs, 1)
                correct_val += (preds == labels).sum().item()
                total_val += labels.size(0)

        val_loss = total_val_loss / len(val_loader)
        val_acc = correct_val / total_val
        val_losses.append(val_loss)
        val_accs.append(val_acc)

        print(f"Epoch {epoch+1}/{n_epochs}: Train Loss: {train_loss:.4f}, Train Acc: {train_acc:.4

    # Plot accuracy and loss curves
    plot_metrics(train_losses, val_losses, train_accs, val_accs)

    return model
```

In [16]:

```
def plot_metrics(train_losses, val_losses, train_accs, val_accs):
    epochs = range(1, len(train_losses) + 1)

    plt.figure(figsize=(12, 5))

    plt.subplot(1, 2, 1)
    plt.plot(epochs, train_losses, label="Train Loss")
    plt.plot(epochs, val_losses, label="Val Loss")
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.legend()
    plt.title("Loss Curve")

    plt.subplot(1, 2, 2)
    plt.plot(epochs, train_accs, label="Train Accuracy")
    plt.plot(epochs, val_accs, label="Val Accuracy")
    plt.xlabel("Epochs")
    plt.ylabel("Accuracy")
    plt.legend()
    plt.title("Accuracy Curve")

    plt.show()
```

In [17]:

```
def test_model(model, test_loader, device):
    model.eval()
    correct, total = 0, 0

    with torch.no_grad():
        for images, labels in test_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            _, preds = torch.max(outputs, 1)
            correct += (preds == labels).sum().item()
            total += labels.size(0)

    accuracy = correct / total
    print(f"Test Accuracy: {accuracy:.4f}")
    return accuracy
```

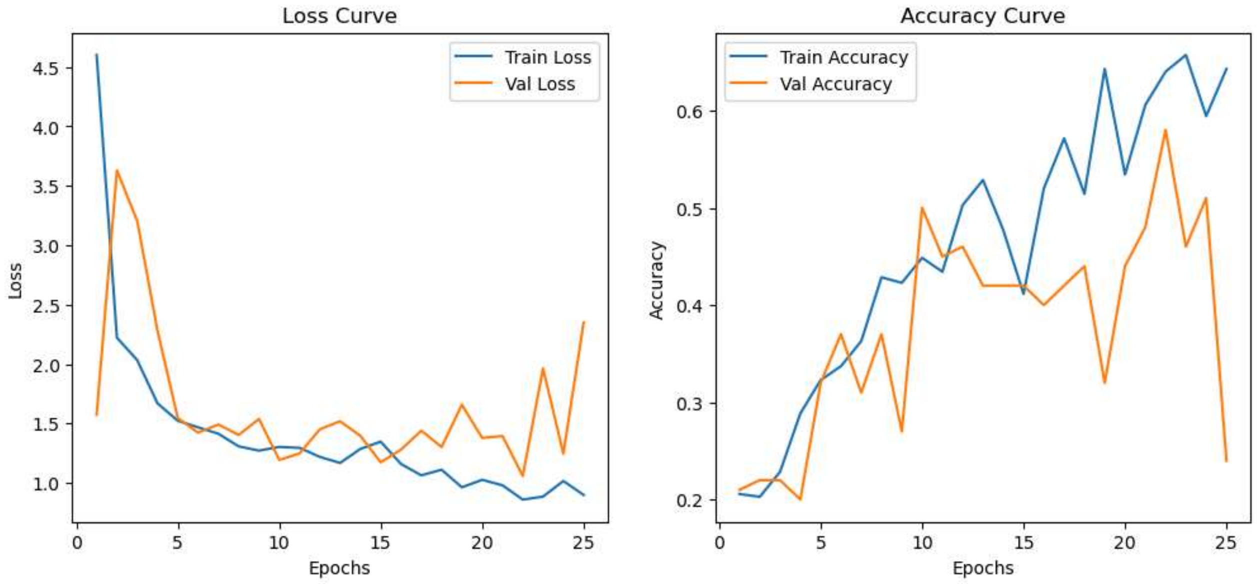
In [18]:

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

In [19]:

```
model = CNN(num_classes)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
n_epochs = 25
model = train_model(model, train_loader, val_loader, criterion, optimizer, n_epochs, device)
```

```
Epoch 1/25: Train Loss: 4.6006, Train Acc: 0.2057, Val Loss: 1.5738, Val Acc: 0.2100
Epoch 2/25: Train Loss: 2.2216, Train Acc: 0.2029, Val Loss: 3.6301, Val Acc: 0.2200
Epoch 3/25: Train Loss: 2.0326, Train Acc: 0.2286, Val Loss: 3.2042, Val Acc: 0.2200
Epoch 4/25: Train Loss: 1.6689, Train Acc: 0.2886, Val Loss: 2.2799, Val Acc: 0.2000
Epoch 5/25: Train Loss: 1.5224, Train Acc: 0.3229, Val Loss: 1.5448, Val Acc: 0.3200
Epoch 6/25: Train Loss: 1.4673, Train Acc: 0.3371, Val Loss: 1.4228, Val Acc: 0.3700
Epoch 7/25: Train Loss: 1.4135, Train Acc: 0.3629, Val Loss: 1.4914, Val Acc: 0.3100
Epoch 8/25: Train Loss: 1.3072, Train Acc: 0.4286, Val Loss: 1.4039, Val Acc: 0.3700
Epoch 9/25: Train Loss: 1.2718, Train Acc: 0.4229, Val Loss: 1.5378, Val Acc: 0.2700
Epoch 10/25: Train Loss: 1.3023, Train Acc: 0.4486, Val Loss: 1.1937, Val Acc: 0.5000
Epoch 11/25: Train Loss: 1.2953, Train Acc: 0.4343, Val Loss: 1.2478, Val Acc: 0.4500
Epoch 12/25: Train Loss: 1.2186, Train Acc: 0.5029, Val Loss: 1.4510, Val Acc: 0.4600
Epoch 13/25: Train Loss: 1.1677, Train Acc: 0.5286, Val Loss: 1.5185, Val Acc: 0.4200
Epoch 14/25: Train Loss: 1.2866, Train Acc: 0.4771, Val Loss: 1.3954, Val Acc: 0.4200
Epoch 15/25: Train Loss: 1.3470, Train Acc: 0.4114, Val Loss: 1.1738, Val Acc: 0.4200
Epoch 16/25: Train Loss: 1.1587, Train Acc: 0.5200, Val Loss: 1.2800, Val Acc: 0.4000
Epoch 17/25: Train Loss: 1.0637, Train Acc: 0.5714, Val Loss: 1.4394, Val Acc: 0.4200
Epoch 18/25: Train Loss: 1.1109, Train Acc: 0.5143, Val Loss: 1.3018, Val Acc: 0.4400
Epoch 19/25: Train Loss: 0.9628, Train Acc: 0.6429, Val Loss: 1.6579, Val Acc: 0.3200
Epoch 20/25: Train Loss: 1.0259, Train Acc: 0.5343, Val Loss: 1.3786, Val Acc: 0.4400
Epoch 21/25: Train Loss: 0.9797, Train Acc: 0.6057, Val Loss: 1.3940, Val Acc: 0.4800
Epoch 22/25: Train Loss: 0.8595, Train Acc: 0.6400, Val Loss: 1.0577, Val Acc: 0.5800
Epoch 23/25: Train Loss: 0.8847, Train Acc: 0.6571, Val Loss: 1.9636, Val Acc: 0.4600
Epoch 24/25: Train Loss: 1.0161, Train Acc: 0.5943, Val Loss: 1.2457, Val Acc: 0.5100
Epoch 25/25: Train Loss: 0.8984, Train Acc: 0.6429, Val Loss: 2.3489, Val Acc: 0.2400
```



In [20]:

```
test_model(model, test_loader, device)
```

```
Test Accuracy: 0.2600
```

Out[20]:

```
0.26
```

In [ ]: