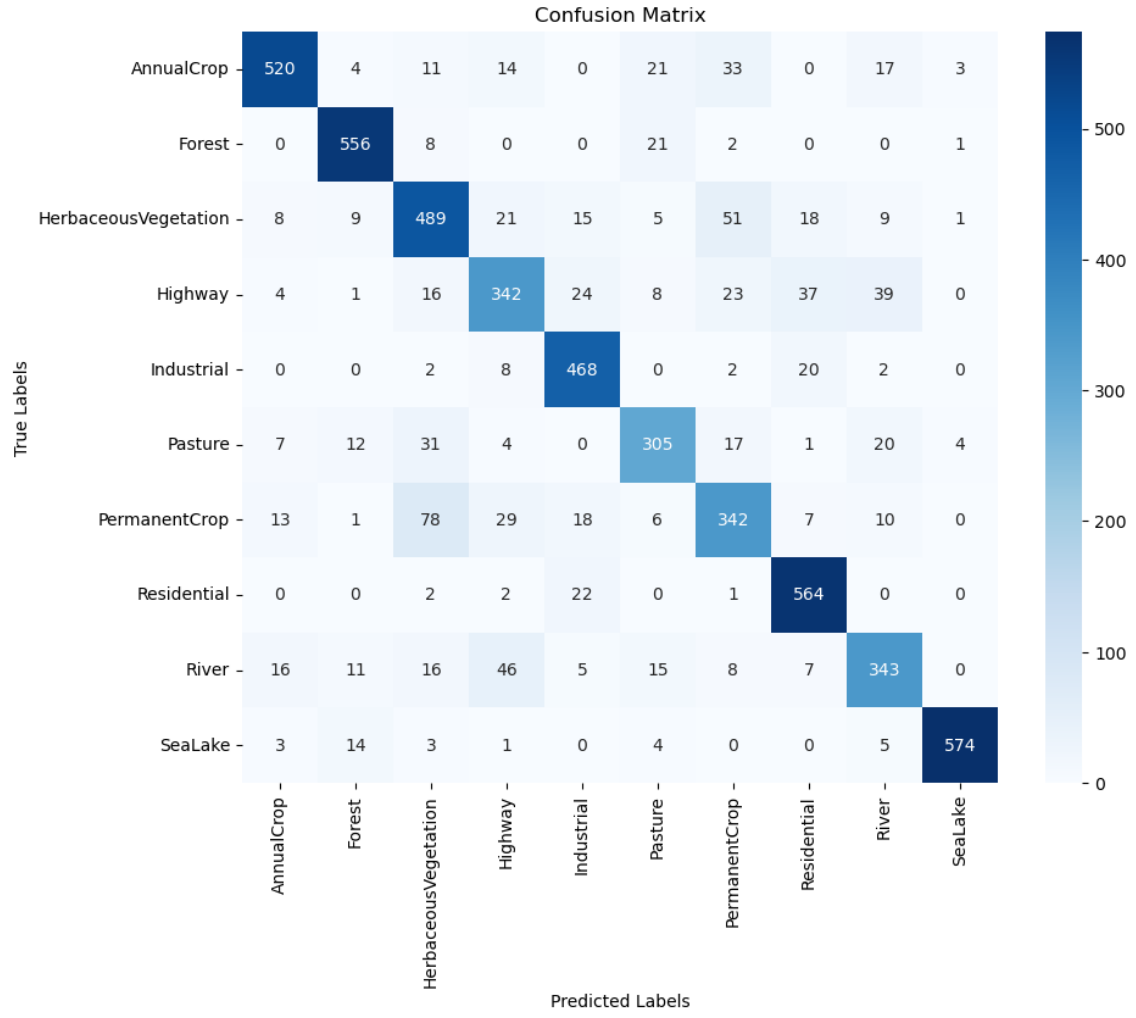
In [3]:

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.transforms as transforms
from torchvision.datasets import ImageFolder
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import confusion_matrix
import seaborn as sns

data_dir = r'C:\Users\anura\Downloads\data\EuroSAT_RGB\MNIST\raw\STA380-master\data\Eurc

# Data Preprocessing
transform = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

dataset = ImageFolder(root=data_dir, transform=transform)

train_size = int(0.8 * len(dataset))
test_size = len(dataset) - train_size

train_set, test_set = torch.utils.data.random_split(dataset, [train_size, test_size])

batch_size = 32

train_loader = DataLoader(train_set, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_set, batch_size=batch_size, shuffle=False)

class CNNClassifier(nn.Module):
    def __init__(self, num_classes):
        super(CNNClassifier, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 16, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(16, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),
        )
        self.classifier = nn.Sequential(
            nn.Linear(32 * 32 * 32, 128),
            nn.ReLU(),
            nn.Linear(128, num_classes)
        )

    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), -1)
        x = self.classifier(x)
        return x

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

In [5]:

```
# Training the Model
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

num_epochs = 10
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

for epoch in range(num_epochs):
    model.train()
    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

    print(f"Epoch [{epoch + 1}/{num_epochs}] - Loss: {loss.item():.4f}")

# Testing and Evaluation
model.eval()
correct = 0
total = 0
all_preds = []
all_labels = []

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

test_accuracy = correct / total
print(f"Test Set Accuracy: {test_accuracy:.2%}")

# Confusion Matrix
cm = confusion_matrix(all_labels, all_preds)
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=dataset.classes, yticklab
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix")
plt.show()
```

```
Epoch [1/10] - Loss: 0.4122
Epoch [2/10] - Loss: 0.6730
Epoch [3/10] - Loss: 0.3708
Epoch [4/10] - Loss: 0.3811
Epoch [5/10] - Loss: 0.0951
Epoch [6/10] - Loss: 0.1156
Epoch [7/10] - Loss: 0.1658
Epoch [8/10] - Loss: 0.1285
Epoch [9/10] - Loss: 0.1096
Epoch [10/10] - Loss: 0.0311
Test Set Accuracy: 83.39%
```




In [ ]:

```
## RESULT

#overall test-set accuracy came out to be 83.39%
```

In [6]:

```
model.eval()
correct = 0
total = 0
all_preds = []
all_labels = []

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

test_accuracy = correct / total
print(f"Test Set Accuracy: {test_accuracy:.2%}")

# Display example images with predictions in different rows
num_display_images = 15
num_rows = 5  # Number of rows for display
num_images_per_row = num_display_images // num_rows

fig, axes = plt.subplots(num_rows, num_images_per_row, figsize=(15, 6))
model.to("cpu")  # Move the model to the CPU for inference

with torch.no_grad():
    for i in range(num_display_images):
        row = i // num_images_per_row
        col = i % num_images_per_row

        image, label = test_set[i]
        image = image.unsqueeze(0)  # Add batch dimension

        output = model(image)
        _, predicted = torch.max(output, 1)

        predicted_class = dataset.classes[predicted.item()]
        actual_class = dataset.classes[label]

        image = image.squeeze(0)  # Remove batch dimension
        image = image.permute(1, 2, 0)  # Reorder dimensions for visualization
        image = (image * 0.229) + 0.485  # Denormalize image

        axes[row, col].imshow(image)
        axes[row, col].set_title(f"Predicted: {predicted_class}\nActual: {actual_class}"
        axes[row, col].axis("off")

plt.tight_layout()
plt.show()
```

```
Test Set Accuracy: 83.39%

Clipping input data to the valid range for imshow with RGB data ([0..1] f
or floats or [0..255] for integers).
```

Predicted: Forest
Actual: Forest

Predicted: Forest
Actual: Forest

Predicted: Forest
Actual: Forest

Predicted: SeaLake
Actual: SeaLake

Predicted: SeaLake
Actual: SeaLake

Predicted: Highway
Actual: Highway

Predicted: Highway
Actual: Highway

Predicted: HerbaceousVegetation
Actual: HerbaceousVegetation

Predicted: PermanentCrop
Actual: PermanentCrop

Predicted: SeaLake
Actual: SeaLake

Predicted: HerbaceousVegetation
Actual: HerbaceousVegetation

Predicted: HerbaceousVegetation
Actual: Pasture

Predicted: HerbaceousVegetation
Actual: HerbaceousVegetation

Predicted: Highway
Actual: Highway

Predicted: River
Actual: River