Epochs = 15

Learning Rate = 0.001

Optimizer = Adam

Batch Size = 64

In [ ]:
```
# Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')

import torch
from torch.utils.data import DataLoader, random_split
from torchvision import datasets, transforms
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import matplotlib.pyplot as plt
import numpy as np

# Define transformations
transform = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.Grayscale(num_output_channels=1),  # If images are RGB, remove this line
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))  # Normalize for grayscale images
])

# Load dataset
dataset = datasets.ImageFolder('/content/drive/My Drive/A-3_and_numbers/', transform=transform)

# Calculate dataset sizes for train, validation, and test splits
train_size = int(0.7 * len(dataset))
val_size = int(0.15 * len(dataset))
test_size = len(dataset) - train_size - val_size

# Split dataset
train_dataset, val_dataset, test_dataset = random_split(dataset, [train_size, val_size, test_size])

# Create data loaders
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=64, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", for
ce_remount=True).
```

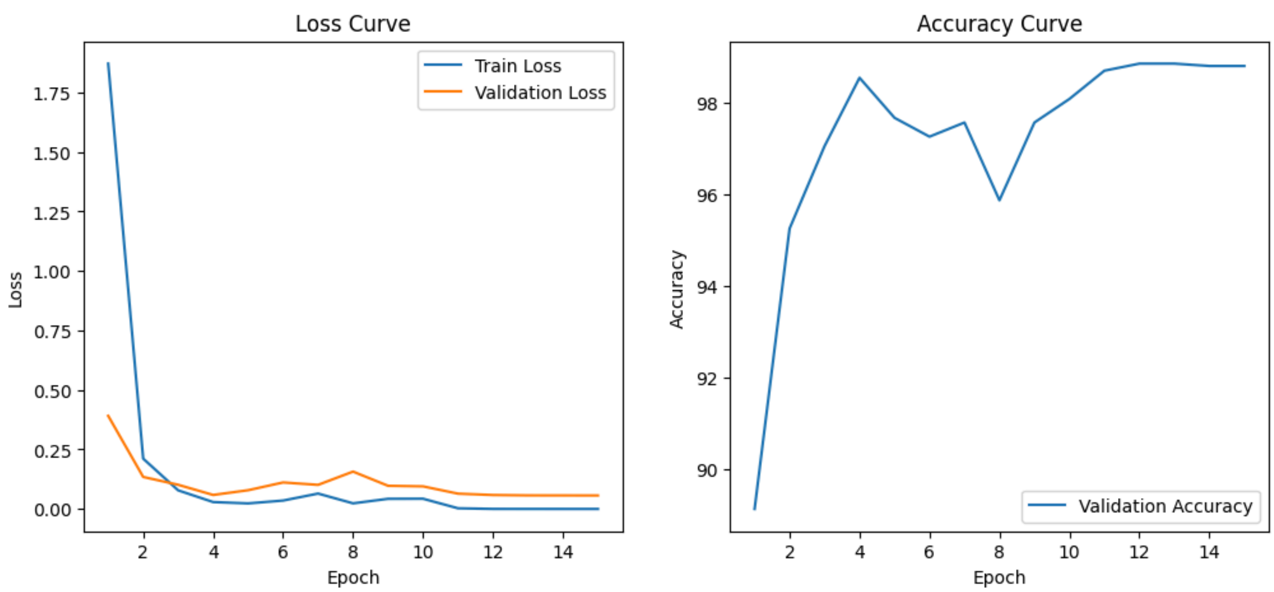
In [ ]:
```
# Define the CNN Model
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()

        self.conv1 = nn.Conv2d(1, 64, kernel_size=3, padding=1)
        self.bn1 = nn.BatchNorm2d(64)
        self.pool1 = nn.MaxPool2d(2, 2)

        self.conv2 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
        self.bn2 = nn.BatchNorm2d(128)
        self.pool2 = nn.MaxPool2d(2, 2)

        self.fc1 = nn.Linear(128 * 32 * 32, 256)  # Adjusted input size
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, 64)
        self.fc4 = nn.Linear(64, len(dataset.classes))

    def forward(self, x):
        x = self.pool1(F.relu(self.bn1(self.conv1(x))))
        x = self.pool2(F.relu(self.bn2(self.conv2(x))))
        x = x.view(x.size(0), -1)  # Flatten the output
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
```

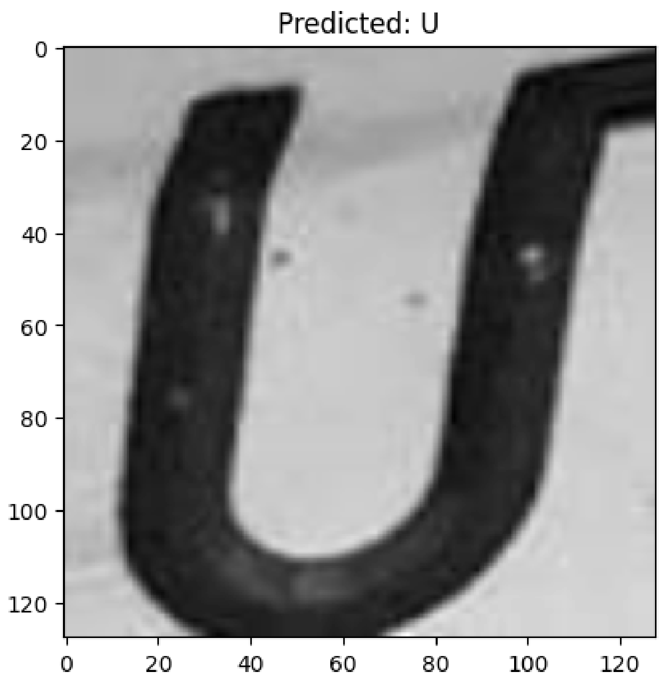
```
        x = F.relu(self.fc3(x))
        x = self.fc4(x)
        return x

model = SimpleCNN()
```

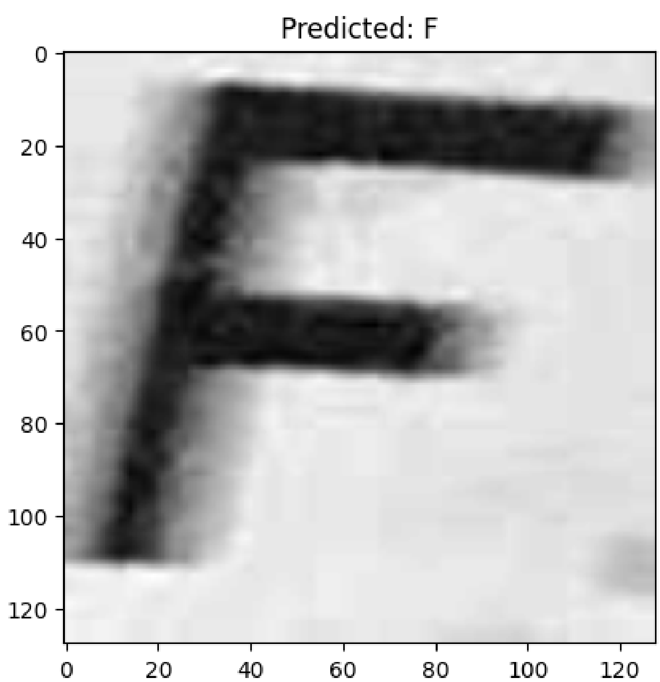
In [ ]:
```
# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Lists to store loss and accuracy values
train_losses = []
val_losses = []
val_accuracies = []

# Training loop
num_epochs = 15

for epoch in range(num_epochs):
    model.train()
    train_loss = 0.0
    correct_train = 0
    total_train = 0

    for inputs, labels in train_loader:
        # Zero the parameter gradients
        optimizer.zero_grad()

        # Forward pass
        outputs = model(inputs)
        loss = criterion(outputs, labels)

        # Backward pass and optimize
        loss.backward()
        optimizer.step()

        train_loss += loss.item() * inputs.size(0)
        _, predicted = torch.max(outputs, 1)
        total_train += labels.size(0)
        correct_train += (predicted == labels).sum().item()

    # Validation
    model.eval()
    val_loss = 0.0
    correct_val = 0
    total_val = 0

    with torch.no_grad():
        for inputs, labels in val_loader:
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            val_loss += loss.item() * inputs.size(0)

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

    train_loss /= len(train_loader.dataset)
    val_loss /= len(val_loader.dataset)
    val_accuracy = 100 * correct_val / total_val

    train_losses.append(train_loss)
    val_losses.append(val_loss)
    val_accuracies.append(val_accuracy)

    print(f'Epoch [{epoch+1}/{num_epochs}], Train Loss: {train_loss:.4f}, Val Loss: {val_loss:.4f}, Val Accuracy

# Plotting the loss and accuracy curves
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(range(1, num_epochs+1), train_losses, label='Train Loss')
plt.plot(range(1, num_epochs+1), val_losses, label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
```

```
plt.title('Loss Curve')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(range(1, num_epochs+1), val_accuracies, label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Accuracy Curve')
plt.legend()

plt.show()

# Function to display images and predictions
def imshow(img, title):
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)), cmap='gray')
    plt.title(title)
    plt.show()

# Predict and display results on all images from the test set
model.eval()
with torch.no_grad():
    for inputs, labels in test_loader:  # Using the test_loader instead of creating a new DataLoader
        outputs = model(inputs)
        _, predicted = torch.max(outputs, 1)

        for i in range(inputs.size(0)):
            predicted_label = dataset.classes[predicted[i].item()]
            imshow(inputs[i], f'Predicted: {predicted_label}')
```

```
Epoch [1/15], Train Loss: 1.8712, Val Loss: 0.3912, Val Accuracy: 89.12%
Epoch [2/15], Train Loss: 0.2115, Val Loss: 0.1346, Val Accuracy: 95.26%
Epoch [3/15], Train Loss: 0.0782, Val Loss: 0.1016, Val Accuracy: 97.06%
Epoch [4/15], Train Loss: 0.0290, Val Loss: 0.0589, Val Accuracy: 98.56%
Epoch [5/15], Train Loss: 0.0234, Val Loss: 0.0787, Val Accuracy: 97.68%
Epoch [6/15], Train Loss: 0.0353, Val Loss: 0.1114, Val Accuracy: 97.27%
Epoch [7/15], Train Loss: 0.0647, Val Loss: 0.1013, Val Accuracy: 97.58%
Epoch [8/15], Train Loss: 0.0236, Val Loss: 0.1570, Val Accuracy: 95.87%
Epoch [9/15], Train Loss: 0.0426, Val Loss: 0.0974, Val Accuracy: 97.58%
Epoch [10/15], Train Loss: 0.0430, Val Loss: 0.0949, Val Accuracy: 98.09%
Epoch [11/15], Train Loss: 0.0029, Val Loss: 0.0643, Val Accuracy: 98.71%
Epoch [12/15], Train Loss: 0.0003, Val Loss: 0.0586, Val Accuracy: 98.87%
Epoch [13/15], Train Loss: 0.0001, Val Loss: 0.0570, Val Accuracy: 98.87%
Epoch [14/15], Train Loss: 0.0001, Val Loss: 0.0569, Val Accuracy: 98.81%
Epoch [15/15], Train Loss: 0.0001, Val Loss: 0.0566, Val Accuracy: 98.81%
```




Predicted: U
0
20
40
60
80
100
120
0 20 40 60 80 100 120


Predicted: F
0
20
40
60
80
100
120
0 20 40 60 80 100 120