```
pip install idx2numpy
```

```
Collecting idx2numpy
  Downloading idx2numpy-1.2.3.tar.gz (6.8 kB)
  Preparing metadata (setup.py) ... ent already satisfied: numpy in
/usr/local/lib/python3.10/dist-packages (from idx2numpy) (1.26.4)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-
packages (from idx2numpy) (1.16.0)
Building wheels for collected packages: idx2numpy
  Building wheel for idx2numpy (setup.py) ... py: filename=idx2numpy-
1.2.3-py3-none-any.whl size=7904
sha256=9bd74dcebf6afcfa6409dde95550773ba0996d64a7d481260c9c0385f170391
a
  Stored in directory:
/root/.cache/pip/wheels/e0/f4/e7/643fc5f932ec2ff92997f43f007660feb23f9
48aa8486f1107
Successfully built idx2numpy
Installing collected packages: idx2numpy
Successfully installed idx2numpy-1.2.3
```

```python
import torch
from torch.utils.data import DataLoader, Subset, Dataset, random_split
from torchvision import datasets, transforms
from sklearn.model_selection import train_test_split
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
import numpy as np
import os
import idx2numpy
import torch.nn as nn
import torch.optim as optim
from sklearn.metrics import confusion_matrix, classification_report
import os
import idx2numpy
RANDOM_SEED = 42
BATCH_SIZE = 64

# File paths
TRAIN_IMAGES_PATH = "/train-images.idx3-ubyte"
TRAIN_LABELS_PATH = "/train-labels.idx1-ubyte"
TEST_IMAGES_PATH = "/t10k-images.idx3-ubyte"
TEST_LABELS_PATH = "/t10k-labels.idx1-ubyte"
# Load images and labels
train_labels = idx2numpy.convert_from_file(TRAIN_LABELS_PATH)
test_labels = idx2numpy.convert_from_file(TEST_LABELS_PATH)
test_images = idx2numpy.convert_from_file(TEST_IMAGES_PATH)
train_images = idx2numpy.convert_from_file(TRAIN_IMAGES_PATH)
# Normalize images to [0, 1] and convert to PyTorch tensors
```

```
transform = transforms.Compose([transforms.ToTensor(),
transforms.Normalize((0.5,), (0.5,))])
# Custom Dataset class
class MNISTDataset(Dataset):
    def __init__(self, images, labels, transform=None): # Changed
_init_ to __init__
        self.images = images
        self.labels = labels
        self.transform = transform
    def __len__(self): # Changed _len_ to __len__
        return len(self.images)
    def __getitem__(self, idx): # Changed _getitem_ to __getitem__
        image = self.images[idx]
        label = self.labels[idx]
        # Convert to float and apply transformations
        if self.transform:
            image = self.transform(image)
        else:
            image = torch.tensor(image, dtype=torch.float32) / 255.0
# Normalize manually
        return image, label
# Create datasets
train_val_dataset = MNISTDataset(train_images, train_labels,
transform=transform)
test_dataset = MNISTDataset(test_images, test_labels,
transform=transform)
# Stratified split for training and validation
# Stratify the split into training (50K) and validation (10K)
indices = np.arange(len(train_val_dataset))
train_indices, val_indices = train_test_split(indices, test_size=1/6,
stratify=train_labels, random_state=RANDOM_SEED)
# Subset datasets
train_dataset = torch.utils.data.Subset(train_val_dataset,
train_indices)
val_dataset = torch.utils.data.Subset(train_val_dataset, val_indices)
# Create DataLoaders
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE,
shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=BATCH_SIZE,
shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE,
shuffle=False)
# Print dataset sizes
print(f"Training dataset size: {len(train_dataset)}")
print(f"Validation dataset size: {len(val_dataset)}")
print(f"Test dataset size: {len(test_dataset)}")
```

```
Training dataset size: 50000
Validation dataset size: 10000
Test dataset size: 10000
```

```python
# Define the feedforward neural network
class FeedforwardNN(nn.Module):  #neural network defined
    def __init__(self, input_size=784, hidden_size1=128, 
hidden_size2=64, output_size=10):
        super(FeedforwardNN, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size1) # make the 
y=w^T x + b
        self.relu1 = nn.ReLU()  # make activation function 1
        self.fc2 = nn.Linear(hidden_size1, hidden_size2)
        self.relu2 = nn.ReLU()  # make activation function 2
        self.fc3 = nn.Linear(hidden_size2, output_size)  # applying 
linearity for output
    def forward(self, x):
        x = x.view(x.size(0), -1)  # Flatten the input  make it 1-d
        x = self.fc1(x)  # Apply the first linear layer
        x = self.relu1(x)  # Apply the activation function
        x = self.fc2(x)  # Apply the second linear layer
        x = self.relu2(x)  # Apply the activation function
        x = self.fc3(x)  # Apply the output linear layer
        return x  # return the output
# Initialize the model, loss function, and optimizer
model = FeedforwardNN()  # make a new neural network
criterion = nn.CrossEntropyLoss()  # Cross-entropy loss
optimizer = optim.SGD(model.parameters(), lr=0.01)  # Gradient 
Stochastic Descent
```

this is our neural network model which is by default composed of (28 x 28) = 784 input units and 2 hidden layers : layer 1 composed of 128 neuron and layer 2 composed of 64 neuron , and output of 10 classes which represents numbers from 0 -> 9
our activation function for layer 1 is relu and for layer 2 is relu and for sure for the output linear

```python
def train_model(model, train_loader, val_loader, criterion, optimizer, 
epochs=100):
    best_val_loss = float('inf')
    patience = 5
    epochs_without_improvement = 0
    train_losses,val_losses,train_acc,val_acc = [], [],[],[]
    for epoch in range(epochs):  # loop for maximum 100 iterations
        model.train() # training the model
        train_loss = 0.0 #initialize loss for each epoch
        for images, labels in train_loader:  # loop on training data
            # Zero the gradients
            optimizer.zero_grad()  # start with zero gradient
            # Forward pass
            outputs = model(images)  # get the expected output of the 
images depending on model
            loss = criterion(outputs, labels)  # calculate the loss 
for the iteration
            # Backward pass
            loss.backward()  # calculate the gradient of the loss
function
            optimizer.step() #make a stochastic gradient descent step
            train_loss += loss.item()
            train_loss /= len(train_loader) # Average training loss
        train_losses.append(train_loss)
        correct = 0
        total = 0
        model.eval()  # Set the model to evaluation mode
        with torch.no_grad():
            for images, labels in train_loader:
                outputs = model(images)
                _, predicted = torch.max(outputs, 1)  # Get predicted
class
                #predicted: stores the indices of the maximum values,
corresponding to the predicted class labels.
                total += labels.size(0) #A counter that accumulates
the number of samples processed so far during the loop.
                correct += (predicted == labels).sum().item()
        trainacc=correct/total
        train_acc.append(trainacc)
        # Validation phase
        model.eval()
        val_loss = 0.0
        with torch.no_grad(): #statement in PyTorch is used to
temporarily disable gradient computation
            for images, labels in val_loader:
                outputs = model(images)
                _, predicted = torch.max(outputs, 1)  # Get predicted
class
                total += labels.size(0)
                correct += (predicted == labels).sum().item()
                loss = criterion(outputs, labels)
                val_loss += loss.item()
                val_loss /= len(val_loader)
            # Average validation loss
        val_losses.append(val_loss)
        validacc=correct/total
        val_acc.append(validacc)
        print(f"Epoch [{epoch+1}/{epochs}], Train Loss:
{train_loss:.4f}, Val Loss: {val_loss:.4f} , train acc :{trainacc:.4f}
, validation acc : {validacc:.4f}")
        #early exit process happen
        if val_loss < best_val_loss:
            best_val_loss = val_loss
            epochs_without_improvement = 0  # Reset patience counter
        else:
            epochs_without_improvement += 1
        if epochs_without_improvement >= patience:
            print("Stopping early!")
            return train_losses, val_losses ,train_acc,val_acc
    return train_losses, val_losses ,train_acc,val_acc
train_losses, val_losses,train_accuracy,val_accuracy =
train_model(model, train_loader, val_loader, criterion, optimizer,
epochs=100)
```

```
/usr/local/lib/python3.10/dist-packages/torchvision/transforms/
functional.py:154: UserWarning: The given NumPy array is not writable,
and PyTorch does not support non-writable tensors. This means writing
to this tensor will result in undefined behavior. You may want to copy
the array to protect its data or make it writable before converting it
to a tensor. This type of warning will be suppressed for the rest of
this program. (Triggered internally at
../torch/csrc/utils/tensor_numpy.cpp:206.)
  img = torch.from_numpy(pic.transpose((2, 0, 1))).contiguous()
```

```
Epoch [1/100], Train Loss: 0.0006, Val Loss: 0.0020 , train
acc :0.8425 , validation acc : 0.8425
Epoch [2/100], Train Loss: 0.0007, Val Loss: 0.0007 , train
acc :0.8948 , validation acc : 0.8946
Epoch [3/100], Train Loss: 0.0002, Val Loss: 0.0004 , train
acc :0.9020 , validation acc : 0.9022
Epoch [4/100], Train Loss: 0.0007, Val Loss: 0.0005 , train
acc :0.8975 , validation acc : 0.8966
Epoch [5/100], Train Loss: 0.0003, Val Loss: 0.0005 , train
acc :0.9142 , validation acc : 0.9135
Epoch [6/100], Train Loss: 0.0003, Val Loss: 0.0003 , train
acc :0.9255 , validation acc : 0.9249
Epoch [7/100], Train Loss: 0.0002, Val Loss: 0.0003 , train
acc :0.9316 , validation acc : 0.9312
Epoch [8/100], Train Loss: 0.0000, Val Loss: 0.0002 , train
acc :0.9366 , validation acc : 0.9358
Epoch [9/100], Train Loss: 0.0009, Val Loss: 0.0003 , train
acc :0.9311 , validation acc : 0.9301
Epoch [10/100], Train Loss: 0.0007, Val Loss: 0.0003 , train
acc :0.9324 , validation acc : 0.9309
Epoch [11/100], Train Loss: 0.0004, Val Loss: 0.0002 , train
acc :0.9443 , validation acc : 0.9431
Epoch [12/100], Train Loss: 0.0001, Val Loss: 0.0001 , train
acc :0.9505 , validation acc : 0.9494
Epoch [13/100], Train Loss: 0.0000, Val Loss: 0.0001 , train
acc :0.9538 , validation acc : 0.9526
Epoch [14/100], Train Loss: 0.0001, Val Loss: 0.0001 , train
acc :0.9552 , validation acc : 0.9538
Epoch [15/100], Train Loss: 0.0009, Val Loss: 0.0001 , train
acc :0.9537 , validation acc : 0.9524
Epoch [16/100], Train Loss: 0.0000, Val Loss: 0.0001 , train
acc :0.9604 , validation acc : 0.9593
Epoch [17/100], Train Loss: 0.0002, Val Loss: 0.0001 , train
```

```
acc :0.9630 , validation acc : 0.9616
Epoch [18/100], Train Loss: 0.0000, Val Loss: 0.0001 , train
acc :0.9657 , validation acc : 0.9641
Epoch [19/100], Train Loss: 0.0006, Val Loss: 0.0001 , train
acc :0.9603 , validation acc : 0.9588
Epoch [20/100], Train Loss: 0.0001, Val Loss: 0.0001 , train
acc :0.9683 , validation acc : 0.9667
Epoch [21/100], Train Loss: 0.0000, Val Loss: 0.0001 , train
acc :0.9659 , validation acc : 0.9644
Epoch [22/100], Train Loss: 0.0000, Val Loss: 0.0001 , train
acc :0.9699 , validation acc : 0.9683
Epoch [23/100], Train Loss: 0.0001, Val Loss: 0.0000 , train
acc :0.9726 , validation acc : 0.9708
Epoch [24/100], Train Loss: 0.0001, Val Loss: 0.0000 , train
acc :0.9709 , validation acc : 0.9689
Epoch [25/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9752 , validation acc : 0.9732
Epoch [26/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9751 , validation acc : 0.9732
Epoch [27/100], Train Loss: 0.0001, Val Loss: 0.0001 , train
acc :0.9741 , validation acc : 0.9722
Epoch [28/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9785 , validation acc : 0.9764
Epoch [29/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9792 , validation acc : 0.9770
Epoch [30/100], Train Loss: 0.0001, Val Loss: 0.0000 , train
acc :0.9800 , validation acc : 0.9777
Epoch [31/100], Train Loss: 0.0003, Val Loss: 0.0000 , train
acc :0.9788 , validation acc : 0.9768
Epoch [32/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9799 , validation acc : 0.9775
Epoch [33/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9822 , validation acc : 0.9798
Epoch [34/100], Train Loss: 0.0002, Val Loss: 0.0000 , train
acc :0.9764 , validation acc : 0.9741
Epoch [35/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9825 , validation acc : 0.9801
Epoch [36/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9835 , validation acc : 0.9809
Epoch [37/100], Train Loss: 0.0009, Val Loss: 0.0001 , train
acc :0.9780 , validation acc : 0.9758
Epoch [38/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9862 , validation acc : 0.9836
Epoch [39/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9843 , validation acc : 0.9819
Epoch [40/100], Train Loss: 0.0001, Val Loss: 0.0000 , train
acc :0.9857 , validation acc : 0.9830
Epoch [41/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9865 , validation acc : 0.9839
```

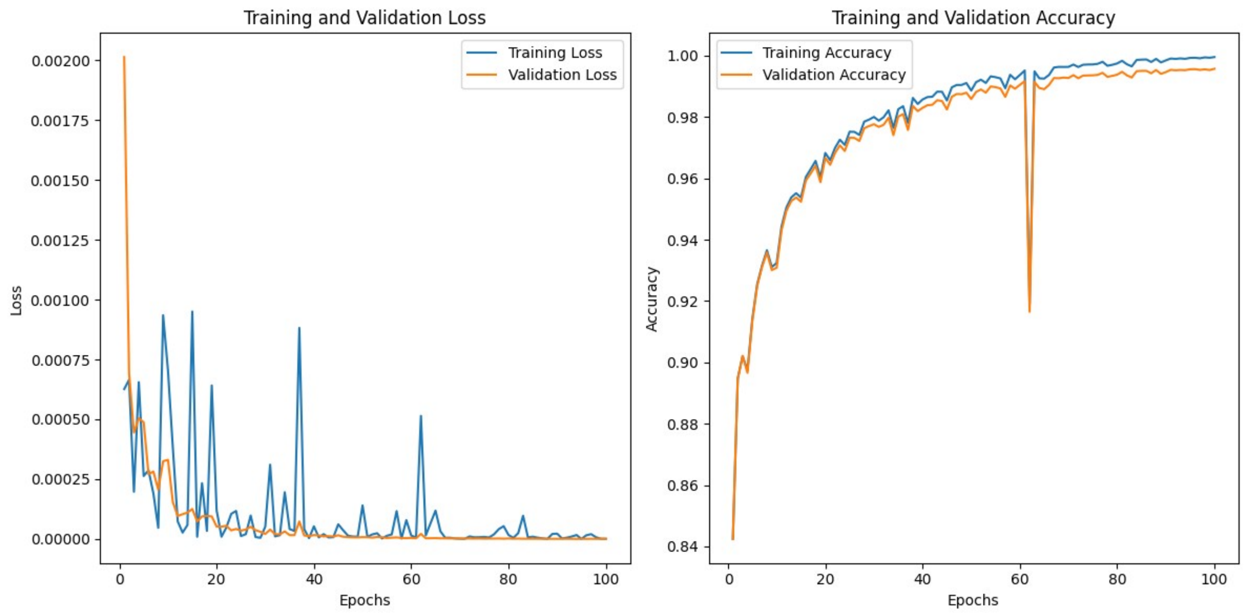
```
Epoch [42/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9866 , validation acc : 0.9840
Epoch [43/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9883 , validation acc : 0.9855
Epoch [44/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9882 , validation acc : 0.9852
Epoch [45/100], Train Loss: 0.0001, Val Loss: 0.0000 , train
acc :0.9854 , validation acc : 0.9825
Epoch [46/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9897 , validation acc : 0.9866
Epoch [47/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9904 , validation acc : 0.9875
Epoch [48/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9904 , validation acc : 0.9874
Epoch [49/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9911 , validation acc : 0.9879
Epoch [50/100], Train Loss: 0.0001, Val Loss: 0.0000 , train
acc :0.9886 , validation acc : 0.9859
Epoch [51/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9914 , validation acc : 0.9882
Epoch [52/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9922 , validation acc : 0.9890
Epoch [53/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9911 , validation acc : 0.9879
Epoch [54/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9933 , validation acc : 0.9900
Epoch [55/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9930 , validation acc : 0.9897
Epoch [56/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9926 , validation acc : 0.9893
Epoch [57/100], Train Loss: 0.0001, Val Loss: 0.0000 , train
acc :0.9894 , validation acc : 0.9866
Epoch [58/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9938 , validation acc : 0.9903
Epoch [59/100], Train Loss: 0.0001, Val Loss: 0.0000 , train
acc :0.9923 , validation acc : 0.9892
Epoch [60/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9937 , validation acc : 0.9905
Epoch [61/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9952 , validation acc : 0.9917
Epoch [62/100], Train Loss: 0.0005, Val Loss: 0.0000 , train
acc :0.9190 , validation acc : 0.9165
Epoch [63/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9949 , validation acc : 0.9914
Epoch [64/100], Train Loss: 0.0001, Val Loss: 0.0000 , train
acc :0.9926 , validation acc : 0.9895
Epoch [65/100], Train Loss: 0.0001, Val Loss: 0.0000 , train
acc :0.9925 , validation acc : 0.9891
Epoch [66/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
```

```
acc :0.9937 , validation acc : 0.9905
Epoch [67/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9961 , validation acc : 0.9927
Epoch [68/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9963 , validation acc : 0.9926
Epoch [69/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9963 , validation acc : 0.9929
Epoch [70/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9963 , validation acc : 0.9927
Epoch [71/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9971 , validation acc : 0.9937
Epoch [72/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9963 , validation acc : 0.9926
Epoch [73/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9970 , validation acc : 0.9935
Epoch [74/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9971 , validation acc : 0.9936
Epoch [75/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9971 , validation acc : 0.9936
Epoch [76/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9973 , validation acc : 0.9937
Epoch [77/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9980 , validation acc : 0.9944
Epoch [78/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9967 , validation acc : 0.9931
Epoch [79/100], Train Loss: 0.0001, Val Loss: 0.0000 , train
acc :0.9970 , validation acc : 0.9934
Epoch [80/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9975 , validation acc : 0.9938
Epoch [81/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9983 , validation acc : 0.9947
Epoch [82/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9972 , validation acc : 0.9936
Epoch [83/100], Train Loss: 0.0001, Val Loss: 0.0000 , train
acc :0.9965 , validation acc : 0.9929
Epoch [84/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9986 , validation acc : 0.9949
Epoch [85/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9987 , validation acc : 0.9950
Epoch [86/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9988 , validation acc : 0.9950
Epoch [87/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9979 , validation acc : 0.9942
Epoch [88/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9989 , validation acc : 0.9953
Epoch [89/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9978 , validation acc : 0.9941
Epoch [90/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9985 , validation acc : 0.9946
```

```
Epoch [91/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9991 , validation acc : 0.9954
Epoch [92/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9989 , validation acc : 0.9952
Epoch [93/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9991 , validation acc : 0.9953
Epoch [94/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9990 , validation acc : 0.9953
Epoch [95/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9993 , validation acc : 0.9956
Epoch [96/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9993 , validation acc : 0.9956
Epoch [97/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9991 , validation acc : 0.9953
Epoch [98/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9994 , validation acc : 0.9956
Epoch [99/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9993 , validation acc : 0.9953
Epoch [100/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9995 , validation acc : 0.9957
```

Applying training for the model and applying stopping early technique

```
def plot_fn():
  correct = 0
  total = 0
  model.eval()  # Set the model to evaluation mode
  with torch.no_grad():
      for images, labels in test_loader:
          outputs = model(images)
          _, predicted = torch.max(outputs, 1)  # Get predicted class
          total += labels.size(0)
          correct += (predicted == labels).sum().item()
  besttest_accuracy=correct/total # till now this is the best test
accuracy
  best_model=model
  print(besttest_accuracy)
  # Calculate accuracies
  # Plot training and validation loss
  epochs = range(1, len(train_losses) + 1)
  plt.figure(figsize=(12, 6))
  # Loss plot
  plt.subplot(1, 2, 1)
  plt.plot(epochs, train_losses, label="Training Loss")
  plt.plot(epochs, val_losses, label="Validation Loss")
  plt.xlabel("Epochs")
  plt.ylabel("Loss")
  plt.title("Training and Validation Loss")
  plt.legend()
  # Accuracy plot
  plt.subplot(1, 2, 2)
  #train_accuracies = [train_accuracy] * len(epochs)  # Placeholder
for per-epoch accuracy
  #val_accuracies = [val_accuracy] * len(epochs)  # Placeholder for
per-epoch accuracy
  plt.plot(epochs, train_accuracy, label="Training Accuracy")
  plt.plot(epochs, val_accuracy, label="Validation Accuracy")
  plt.xlabel("Epochs")
  plt.ylabel("Accuracy")
  plt.title("Training and Validation Accuracy")
  plt.legend()
  plt.tight_layout()
  plt.show()
plot_fn()

0.9771
```



plotting the output of the default neural netwoork and comparing the validation accuracy to the training accuracy and validation error to training error

default model is the one before now we start in analysis ...

Here we are trying 4 different learning rates and making a comparison between them and among them we will choose the best test accuracy which will considerate which model is the best:

```
besttest_accuracy=0.9416
model = FeedforwardNN()  # make a new neural network
criterion = nn.CrossEntropyLoss()  # Cross-entropy loss
optimizer1 = optim.SGD(model.parameters(), lr=0.001)
```

```
train_loss1,val_loss1,train_acc1,val_acc1=train_model(model,
train_loader, val_loader, criterion, optimizer1, epochs=100)
correct = 0
total = 0
model.eval()  # Set the model to evaluation mode
with torch.no_grad():
    for images, labels in test_loader:
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)  # Get predicted class
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
test_accuracy=correct/total # till now this is the best test accuracy
print(f"training with learning rate = 0.001 gives accuracy=
{test_accuracy:.4f}")
if test_accuracy>besttest_accuracy:
  best_model=model
  besttest_accuracy=test_accuracy
#----------------------------------------------------------------------
----------
model = FeedforwardNN()  # make a new neural network
criterion = nn.CrossEntropyLoss()  # Cross-entropy loss
optimizer2 = optim.SGD(model.parameters(), lr=0.0001)
train_loss2,val_loss2,train_acc2,val_acc2=train_model(model,
train_loader, val_loader, criterion, optimizer2, epochs=100)
correct = 0
total = 0
model.eval()  # Set the model to evaluation mode
with torch.no_grad():
    for images, labels in test_loader:
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)  # Get predicted class
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
test_accuracy=correct/total # till now this is the best test accuracy
print(f"training with learning rate = 0.0001 gives accuracy=
{test_accuracy:.4f}")
if test_accuracy>besttest_accuracy:
  best_model=model
  besttest_accuracy=test_accuracy
#----------------------------------------------------------------------
----------
model = FeedforwardNN()  # make a new neural network
criterion = nn.CrossEntropyLoss()  # Cross-entropy loss
optimizer3 = optim.SGD(model.parameters(), lr=0.1)
train_loss3,val_loss3,train_acc3,val_acc3=train_model(model,
train_loader, val_loader, criterion, optimizer3, epochs=100)
correct = 0
total = 0
model.eval()  # Set the model to evaluation mode
```

```
with torch.no_grad():
    for images, labels in test_loader:
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)  # Get predicted class
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
test_accuracy=correct/total # till now this is the best test accuracy
print(f"training with learning rate = 0.1 gives accuracy=
{test_accuracy:.4f}")
if test_accuracy>besttest_accuracy:
  best_model=model
  besttest_accuracy=test_accuracy
#----------------------------------------------------------------------
----------
model = FeedforwardNN()  # make a new neural network
criterion = nn.CrossEntropyLoss()  # Cross-entropy loss
optimizer4 = optim.SGD(model.parameters(), lr=0.05)
train_loss4,val_loss4,train_acc4,val_acc4=train_model(model,
train_loader, val_loader, criterion, optimizer4, epochs=100)
correct = 0
total = 0
model.eval()  # Set the model to evaluation mode
with torch.no_grad():
    for images, labels in test_loader:
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)  # Get predicted class
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
test_accuracy=correct/total # till now this is the best test accuracy
print(f"training with learning rate = 0.05 gives accuracy=
{test_accuracy:.4f}")
if test_accuracy>besttest_accuracy:
  best_model=model
  besttest_accuracy=test_accuracy
#----------------------------------------------------------------------
----------
```

```
Epoch [1/100], Train Loss: 0.0027, Val Loss: 0.0138 , train
acc :0.4940 , validation acc : 0.4933
Epoch [2/100], Train Loss: 0.0024, Val Loss: 0.0118 , train
acc :0.6029 , validation acc : 0.6021
Epoch [3/100], Train Loss: 0.0019, Val Loss: 0.0086 , train
acc :0.6901 , validation acc : 0.6901
Epoch [4/100], Train Loss: 0.0011, Val Loss: 0.0057 , train
acc :0.7542 , validation acc : 0.7532
Epoch [5/100], Train Loss: 0.0010, Val Loss: 0.0040 , train
acc :0.7931 , validation acc : 0.7922
Epoch [6/100], Train Loss: 0.0009, Val Loss: 0.0032 , train
acc :0.8154 , validation acc : 0.8148
Epoch [7/100], Train Loss: 0.0012, Val Loss: 0.0025 , train
```

```
acc :0.8330 , validation acc : 0.8323
Epoch [8/100], Train Loss: 0.0007, Val Loss: 0.0021 , train
acc :0.8471 , validation acc : 0.8463
Epoch [9/100], Train Loss: 0.0004, Val Loss: 0.0018 , train
acc :0.8592 , validation acc : 0.8584
Epoch [10/100], Train Loss: 0.0005, Val Loss: 0.0016 , train
acc :0.8681 , validation acc : 0.8673
Epoch [11/100], Train Loss: 0.0004, Val Loss: 0.0014 , train
acc :0.8749 , validation acc : 0.8742
Epoch [12/100], Train Loss: 0.0007, Val Loss: 0.0012 , train
acc :0.8800 , validation acc : 0.8794
Epoch [13/100], Train Loss: 0.0002, Val Loss: 0.0011 , train
acc :0.8838 , validation acc : 0.8831
Epoch [14/100], Train Loss: 0.0004, Val Loss: 0.0009 , train
acc :0.8880 , validation acc : 0.8875
Epoch [15/100], Train Loss: 0.0001, Val Loss: 0.0009 , train
acc :0.8897 , validation acc : 0.8890
Epoch [16/100], Train Loss: 0.0004, Val Loss: 0.0008 , train
acc :0.8927 , validation acc : 0.8922
Epoch [17/100], Train Loss: 0.0002, Val Loss: 0.0007 , train
acc :0.8932 , validation acc : 0.8928
Epoch [18/100], Train Loss: 0.0006, Val Loss: 0.0007 , train
acc :0.8955 , validation acc : 0.8951
Epoch [19/100], Train Loss: 0.0004, Val Loss: 0.0006 , train
acc :0.8976 , validation acc : 0.8972
Epoch [20/100], Train Loss: 0.0003, Val Loss: 0.0006 , train
acc :0.8996 , validation acc : 0.8993
Epoch [21/100], Train Loss: 0.0004, Val Loss: 0.0006 , train
acc :0.9005 , validation acc : 0.9000
Epoch [22/100], Train Loss: 0.0004, Val Loss: 0.0005 , train
acc :0.9026 , validation acc : 0.9022
Epoch [23/100], Train Loss: 0.0001, Val Loss: 0.0005 , train
acc :0.9040 , validation acc : 0.9035
Epoch [24/100], Train Loss: 0.0007, Val Loss: 0.0005 , train
acc :0.9045 , validation acc : 0.9042
Epoch [25/100], Train Loss: 0.0005, Val Loss: 0.0005 , train
acc :0.9049 , validation acc : 0.9045
Epoch [26/100], Train Loss: 0.0002, Val Loss: 0.0004 , train
acc :0.9064 , validation acc : 0.9059
Epoch [27/100], Train Loss: 0.0011, Val Loss: 0.0004 , train
acc :0.9074 , validation acc : 0.9069
Epoch [28/100], Train Loss: 0.0003, Val Loss: 0.0004 , train
acc :0.9077 , validation acc : 0.9072
Epoch [29/100], Train Loss: 0.0001, Val Loss: 0.0004 , train
acc :0.9088 , validation acc : 0.9082
Epoch [30/100], Train Loss: 0.0002, Val Loss: 0.0004 , train
acc :0.9097 , validation acc : 0.9091
Epoch [31/100], Train Loss: 0.0002, Val Loss: 0.0004 , train
acc :0.9111 , validation acc : 0.9106
```

```
Epoch [32/100], Train Loss: 0.0004, Val Loss: 0.0004 , train
acc :0.9110 , validation acc : 0.9104
Epoch [33/100], Train Loss: 0.0002, Val Loss: 0.0004 , train
acc :0.9123 , validation acc : 0.9120
Epoch [34/100], Train Loss: 0.0005, Val Loss: 0.0004 , train
acc :0.9127 , validation acc : 0.9121
Epoch [35/100], Train Loss: 0.0003, Val Loss: 0.0003 , train
acc :0.9139 , validation acc : 0.9133
Epoch [36/100], Train Loss: 0.0004, Val Loss: 0.0003 , train
acc :0.9141 , validation acc : 0.9136
Epoch [37/100], Train Loss: 0.0005, Val Loss: 0.0003 , train
acc :0.9147 , validation acc : 0.9144
Epoch [38/100], Train Loss: 0.0003, Val Loss: 0.0003 , train
acc :0.9151 , validation acc : 0.9146
Epoch [39/100], Train Loss: 0.0004, Val Loss: 0.0003 , train
acc :0.9162 , validation acc : 0.9158
Epoch [40/100], Train Loss: 0.0001, Val Loss: 0.0003 , train
acc :0.9164 , validation acc : 0.9159
Epoch [41/100], Train Loss: 0.0002, Val Loss: 0.0003 , train
acc :0.9167 , validation acc : 0.9161
Epoch [42/100], Train Loss: 0.0006, Val Loss: 0.0003 , train
acc :0.9187 , validation acc : 0.9182
Epoch [43/100], Train Loss: 0.0001, Val Loss: 0.0003 , train
acc :0.9188 , validation acc : 0.9180
Epoch [44/100], Train Loss: 0.0003, Val Loss: 0.0003 , train
acc :0.9182 , validation acc : 0.9175
Epoch [45/100], Train Loss: 0.0002, Val Loss: 0.0003 , train
acc :0.9205 , validation acc : 0.9198
Epoch [46/100], Train Loss: 0.0005, Val Loss: 0.0003 , train
acc :0.9196 , validation acc : 0.9191
Epoch [47/100], Train Loss: 0.0000, Val Loss: 0.0003 , train
acc :0.9221 , validation acc : 0.9213
Epoch [48/100], Train Loss: 0.0006, Val Loss: 0.0003 , train
acc :0.9219 , validation acc : 0.9211
Stopping early!
training with learning rate = 0.001 gives accuracy= 0.9230
Epoch [1/100], Train Loss: 0.0029, Val Loss: 0.0150 , train
acc :0.1133 , validation acc : 0.1132
Epoch [2/100], Train Loss: 0.0029, Val Loss: 0.0150 , train
acc :0.1162 , validation acc : 0.1160
Epoch [3/100], Train Loss: 0.0030, Val Loss: 0.0149 , train
acc :0.1210 , validation acc : 0.1206
Epoch [4/100], Train Loss: 0.0029, Val Loss: 0.0148 , train
acc :0.1281 , validation acc : 0.1276
Epoch [5/100], Train Loss: 0.0029, Val Loss: 0.0148 , train
acc :0.1380 , validation acc : 0.1373
Epoch [6/100], Train Loss: 0.0028, Val Loss: 0.0147 , train
acc :0.1504 , validation acc : 0.1494
Epoch [7/100], Train Loss: 0.0029, Val Loss: 0.0146 , train
```

```
acc :0.1643 , validation acc : 0.1637
Epoch [8/100], Train Loss: 0.0028, Val Loss: 0.0146 , train
acc :0.1787 , validation acc : 0.1778
Epoch [9/100], Train Loss: 0.0029, Val Loss: 0.0145 , train
acc :0.1935 , validation acc : 0.1928
Epoch [10/100], Train Loss: 0.0028, Val Loss: 0.0144 , train
acc :0.2102 , validation acc : 0.2092
Epoch [11/100], Train Loss: 0.0028, Val Loss: 0.0144 , train
acc :0.2255 , validation acc : 0.2247
Epoch [12/100], Train Loss: 0.0028, Val Loss: 0.0143 , train
acc :0.2440 , validation acc : 0.2434
Epoch [13/100], Train Loss: 0.0028, Val Loss: 0.0142 , train
acc :0.2637 , validation acc : 0.2632
Epoch [14/100], Train Loss: 0.0027, Val Loss: 0.0141 , train
acc :0.2851 , validation acc : 0.2840
Epoch [15/100], Train Loss: 0.0028, Val Loss: 0.0140 , train
acc :0.3040 , validation acc : 0.3035
Epoch [16/100], Train Loss: 0.0027, Val Loss: 0.0139 , train
acc :0.3269 , validation acc : 0.3265
Epoch [17/100], Train Loss: 0.0027, Val Loss: 0.0138 , train
acc :0.3472 , validation acc : 0.3470
Epoch [18/100], Train Loss: 0.0027, Val Loss: 0.0136 , train
acc :0.3679 , validation acc : 0.3677
Epoch [19/100], Train Loss: 0.0027, Val Loss: 0.0135 , train
acc :0.3861 , validation acc : 0.3858
Epoch [20/100], Train Loss: 0.0027, Val Loss: 0.0133 , train
acc :0.4037 , validation acc : 0.4032
Epoch [21/100], Train Loss: 0.0025, Val Loss: 0.0132 , train
acc :0.4190 , validation acc : 0.4188
Epoch [22/100], Train Loss: 0.0026, Val Loss: 0.0130 , train
acc :0.4352 , validation acc : 0.4350
Epoch [23/100], Train Loss: 0.0025, Val Loss: 0.0128 , train
acc :0.4489 , validation acc : 0.4486
Epoch [24/100], Train Loss: 0.0024, Val Loss: 0.0126 , train
acc :0.4626 , validation acc : 0.4621
Epoch [25/100], Train Loss: 0.0024, Val Loss: 0.0123 , train
acc :0.4762 , validation acc : 0.4760
Epoch [26/100], Train Loss: 0.0024, Val Loss: 0.0121 , train
acc :0.4897 , validation acc : 0.4892
Epoch [27/100], Train Loss: 0.0023, Val Loss: 0.0118 , train
acc :0.4999 , validation acc : 0.4993
Epoch [28/100], Train Loss: 0.0024, Val Loss: 0.0116 , train
acc :0.5110 , validation acc : 0.5103
Epoch [29/100], Train Loss: 0.0023, Val Loss: 0.0113 , train
acc :0.5247 , validation acc : 0.5239
Epoch [30/100], Train Loss: 0.0024, Val Loss: 0.0110 , train
acc :0.5359 , validation acc : 0.5351
Epoch [31/100], Train Loss: 0.0022, Val Loss: 0.0106 , train
acc :0.5486 , validation acc : 0.5476
```

```
Epoch [32/100], Train Loss: 0.0022, Val Loss: 0.0103 , train
acc :0.5613 , validation acc : 0.5603
Epoch [33/100], Train Loss: 0.0019, Val Loss: 0.0100 , train
acc :0.5743 , validation acc : 0.5733
Epoch [34/100], Train Loss: 0.0020, Val Loss: 0.0097 , train
acc :0.5883 , validation acc : 0.5870
Epoch [35/100], Train Loss: 0.0017, Val Loss: 0.0093 , train
acc :0.6031 , validation acc : 0.6021
Epoch [36/100], Train Loss: 0.0021, Val Loss: 0.0090 , train
acc :0.6176 , validation acc : 0.6168
Epoch [37/100], Train Loss: 0.0018, Val Loss: 0.0087 , train
acc :0.6308 , validation acc : 0.6301
Epoch [38/100], Train Loss: 0.0021, Val Loss: 0.0084 , train
acc :0.6464 , validation acc : 0.6460
Epoch [39/100], Train Loss: 0.0015, Val Loss: 0.0081 , train
acc :0.6582 , validation acc : 0.6580
Epoch [40/100], Train Loss: 0.0017, Val Loss: 0.0078 , train
acc :0.6694 , validation acc : 0.6694
Epoch [41/100], Train Loss: 0.0017, Val Loss: 0.0075 , train
acc :0.6801 , validation acc : 0.6804
Epoch [42/100], Train Loss: 0.0012, Val Loss: 0.0072 , train
acc :0.6910 , validation acc : 0.6911
Epoch [43/100], Train Loss: 0.0016, Val Loss: 0.0070 , train
acc :0.7013 , validation acc : 0.7013
Epoch [44/100], Train Loss: 0.0018, Val Loss: 0.0067 , train
acc :0.7105 , validation acc : 0.7104
Epoch [45/100], Train Loss: 0.0017, Val Loss: 0.0065 , train
acc :0.7171 , validation acc : 0.7170
Epoch [46/100], Train Loss: 0.0013, Val Loss: 0.0063 , train
acc :0.7245 , validation acc : 0.7243
Epoch [47/100], Train Loss: 0.0014, Val Loss: 0.0061 , train
acc :0.7333 , validation acc : 0.7330
Epoch [48/100], Train Loss: 0.0014, Val Loss: 0.0058 , train
acc :0.7412 , validation acc : 0.7410
Epoch [49/100], Train Loss: 0.0016, Val Loss: 0.0057 , train
acc :0.7491 , validation acc : 0.7487
Epoch [50/100], Train Loss: 0.0016, Val Loss: 0.0055 , train
acc :0.7554 , validation acc : 0.7548
Epoch [51/100], Train Loss: 0.0011, Val Loss: 0.0053 , train
acc :0.7607 , validation acc : 0.7602
Epoch [52/100], Train Loss: 0.0012, Val Loss: 0.0052 , train
acc :0.7667 , validation acc : 0.7662
Epoch [53/100], Train Loss: 0.0013, Val Loss: 0.0050 , train
acc :0.7721 , validation acc : 0.7718
Epoch [54/100], Train Loss: 0.0013, Val Loss: 0.0048 , train
acc :0.7767 , validation acc : 0.7762
Epoch [55/100], Train Loss: 0.0012, Val Loss: 0.0047 , train
acc :0.7812 , validation acc : 0.7809
Epoch [56/100], Train Loss: 0.0008, Val Loss: 0.0046 , train
```

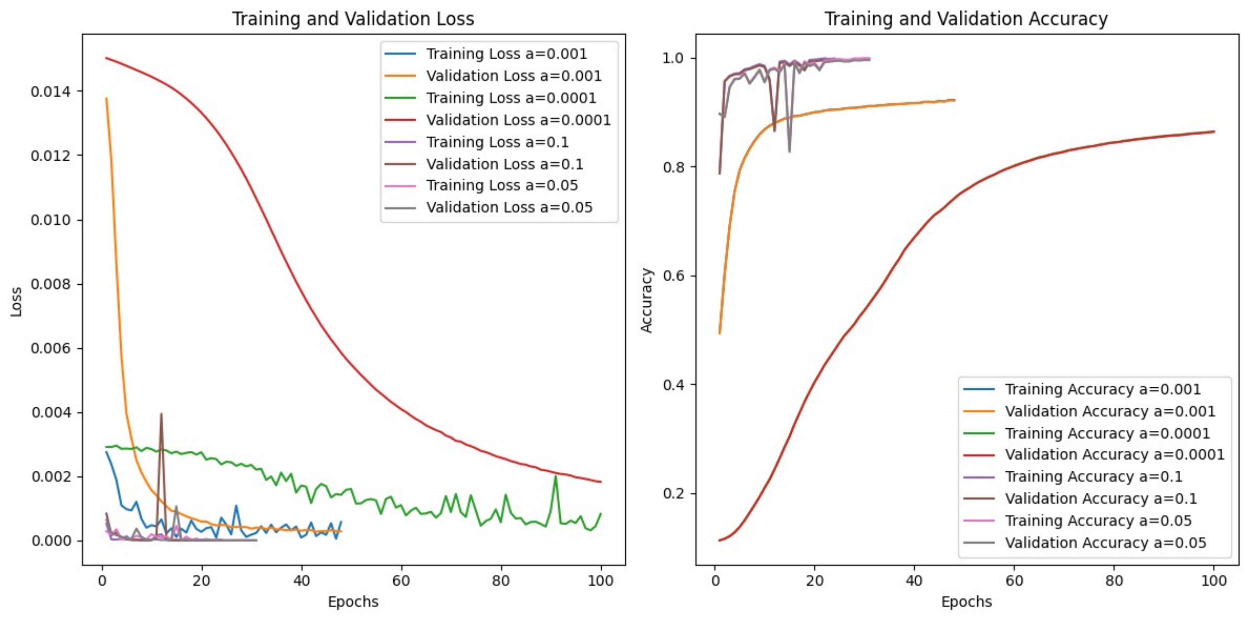
```
acc :0.7851 , validation acc : 0.7845
Epoch [57/100], Train Loss: 0.0013, Val Loss: 0.0044 , train
acc :0.7899 , validation acc : 0.7894
Epoch [58/100], Train Loss: 0.0014, Val Loss: 0.0043 , train
acc :0.7940 , validation acc : 0.7935
Epoch [59/100], Train Loss: 0.0010, Val Loss: 0.0042 , train
acc :0.7978 , validation acc : 0.7970
Epoch [60/100], Train Loss: 0.0011, Val Loss: 0.0041 , train
acc :0.8013 , validation acc : 0.8005
Epoch [61/100], Train Loss: 0.0008, Val Loss: 0.0040 , train
acc :0.8048 , validation acc : 0.8039
Epoch [62/100], Train Loss: 0.0009, Val Loss: 0.0039 , train
acc :0.8081 , validation acc : 0.8073
Epoch [63/100], Train Loss: 0.0010, Val Loss: 0.0038 , train
acc :0.8107 , validation acc : 0.8098
Epoch [64/100], Train Loss: 0.0008, Val Loss: 0.0037 , train
acc :0.8140 , validation acc : 0.8131
Epoch [65/100], Train Loss: 0.0008, Val Loss: 0.0036 , train
acc :0.8168 , validation acc : 0.8160
Epoch [66/100], Train Loss: 0.0009, Val Loss: 0.0035 , train
acc :0.8188 , validation acc : 0.8179
Epoch [67/100], Train Loss: 0.0007, Val Loss: 0.0034 , train
acc :0.8214 , validation acc : 0.8206
Epoch [68/100], Train Loss: 0.0008, Val Loss: 0.0034 , train
acc :0.8238 , validation acc : 0.8229
Epoch [69/100], Train Loss: 0.0014, Val Loss: 0.0033 , train
acc :0.8254 , validation acc : 0.8246
Epoch [70/100], Train Loss: 0.0009, Val Loss: 0.0032 , train
acc :0.8276 , validation acc : 0.8266
Epoch [71/100], Train Loss: 0.0014, Val Loss: 0.0031 , train
acc :0.8301 , validation acc : 0.8292
Epoch [72/100], Train Loss: 0.0009, Val Loss: 0.0031 , train
acc :0.8317 , validation acc : 0.8309
Epoch [73/100], Train Loss: 0.0007, Val Loss: 0.0030 , train
acc :0.8335 , validation acc : 0.8327
Epoch [74/100], Train Loss: 0.0014, Val Loss: 0.0029 , train
acc :0.8352 , validation acc : 0.8344
Epoch [75/100], Train Loss: 0.0009, Val Loss: 0.0029 , train
acc :0.8366 , validation acc : 0.8360
Epoch [76/100], Train Loss: 0.0005, Val Loss: 0.0028 , train
acc :0.8378 , validation acc : 0.8373
Epoch [77/100], Train Loss: 0.0006, Val Loss: 0.0027 , train
acc :0.8399 , validation acc : 0.8392
Epoch [78/100], Train Loss: 0.0007, Val Loss: 0.0027 , train
acc :0.8414 , validation acc : 0.8407
Epoch [79/100], Train Loss: 0.0010, Val Loss: 0.0026 , train
acc :0.8433 , validation acc : 0.8427
Epoch [80/100], Train Loss: 0.0006, Val Loss: 0.0026 , train
acc :0.8444 , validation acc : 0.8437
```

```
Epoch [81/100], Train Loss: 0.0014, Val Loss: 0.0025 , train
acc :0.8453 , validation acc : 0.8446
Epoch [82/100], Train Loss: 0.0009, Val Loss: 0.0025 , train
acc :0.8472 , validation acc : 0.8465
Epoch [83/100], Train Loss: 0.0007, Val Loss: 0.0024 , train
acc :0.8481 , validation acc : 0.8474
Epoch [84/100], Train Loss: 0.0007, Val Loss: 0.0024 , train
acc :0.8497 , validation acc : 0.8489
Epoch [85/100], Train Loss: 0.0005, Val Loss: 0.0024 , train
acc :0.8508 , validation acc : 0.8500
Epoch [86/100], Train Loss: 0.0006, Val Loss: 0.0023 , train
acc :0.8519 , validation acc : 0.8510
Epoch [87/100], Train Loss: 0.0006, Val Loss: 0.0023 , train
acc :0.8529 , validation acc : 0.8519
Epoch [88/100], Train Loss: 0.0005, Val Loss: 0.0022 , train
acc :0.8538 , validation acc : 0.8529
Epoch [89/100], Train Loss: 0.0004, Val Loss: 0.0022 , train
acc :0.8551 , validation acc : 0.8540
Epoch [90/100], Train Loss: 0.0009, Val Loss: 0.0021 , train
acc :0.8560 , validation acc : 0.8549
Epoch [91/100], Train Loss: 0.0020, Val Loss: 0.0021 , train
acc :0.8568 , validation acc : 0.8559
Epoch [92/100], Train Loss: 0.0005, Val Loss: 0.0021 , train
acc :0.8575 , validation acc : 0.8567
Epoch [93/100], Train Loss: 0.0005, Val Loss: 0.0021 , train
acc :0.8580 , validation acc : 0.8571
Epoch [94/100], Train Loss: 0.0006, Val Loss: 0.0020 , train
acc :0.8591 , validation acc : 0.8581
Epoch [95/100], Train Loss: 0.0005, Val Loss: 0.0020 , train
acc :0.8602 , validation acc : 0.8592
Epoch [96/100], Train Loss: 0.0008, Val Loss: 0.0019 , train
acc :0.8611 , validation acc : 0.8602
Epoch [97/100], Train Loss: 0.0004, Val Loss: 0.0019 , train
acc :0.8617 , validation acc : 0.8608
Epoch [98/100], Train Loss: 0.0003, Val Loss: 0.0019 , train
acc :0.8625 , validation acc : 0.8616
Epoch [99/100], Train Loss: 0.0004, Val Loss: 0.0018 , train
acc :0.8633 , validation acc : 0.8625
Epoch [100/100], Train Loss: 0.0008, Val Loss: 0.0018 , train
acc :0.8643 , validation acc : 0.8634
training with learning rate = 0.0001 gives accuracy= 0.8665
Epoch [1/100], Train Loss: 0.0005, Val Loss: 0.0008 , train
acc :0.7879 , validation acc : 0.7868
Epoch [2/100], Train Loss: 0.0000, Val Loss: 0.0002 , train
acc :0.9570 , validation acc : 0.9560
Epoch [3/100], Train Loss: 0.0000, Val Loss: 0.0002 , train
acc :0.9665 , validation acc : 0.9652
Epoch [4/100], Train Loss: 0.0001, Val Loss: 0.0001 , train
acc :0.9706 , validation acc : 0.9691
```

```
Epoch [5/100], Train Loss: 0.0001, Val Loss: 0.0000 , train
acc :0.9714 , validation acc : 0.9696
Epoch [6/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9787 , validation acc : 0.9764
Epoch [7/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9805 , validation acc : 0.9786
Epoch [8/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9845 , validation acc : 0.9821
Epoch [9/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9878 , validation acc : 0.9855
Epoch [10/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9848 , validation acc : 0.9824
Epoch [11/100], Train Loss: 0.0002, Val Loss: 0.0000 , train
acc :0.9610 , validation acc : 0.9585
Epoch [12/100], Train Loss: 0.0002, Val Loss: 0.0039 , train
acc :0.8659 , validation acc : 0.8648
Epoch [13/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9924 , validation acc : 0.9898
Epoch [14/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9942 , validation acc : 0.9912
Epoch [15/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9875 , validation acc : 0.9844
Epoch [16/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9944 , validation acc : 0.9912
Epoch [17/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9886 , validation acc : 0.9855
Epoch [18/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9788 , validation acc : 0.9762
Epoch [19/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9956 , validation acc : 0.9924
Epoch [20/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9962 , validation acc : 0.9931
Epoch [21/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9973 , validation acc : 0.9941
Epoch [22/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9986 , validation acc : 0.9952
Epoch [23/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9973 , validation acc : 0.9940
Epoch [24/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9984 , validation acc : 0.9950
Stopping early!
training with learning rate = 0.1 gives accuracy= 0.9785
Epoch [1/100], Train Loss: 0.0003, Val Loss: 0.0006 , train
acc :0.8970 , validation acc : 0.8964
Epoch [2/100], Train Loss: 0.0002, Val Loss: 0.0002 , train
acc :0.8914 , validation acc : 0.8903
Epoch [3/100], Train Loss: 0.0003, Val Loss: 0.0002 , train
acc :0.9465 , validation acc : 0.9451
Epoch [4/100], Train Loss: 0.0000, Val Loss: 0.0001 , train
```

```
acc :0.9612 , validation acc : 0.9600
Epoch [5/100], Train Loss: 0.0001, Val Loss: 0.0001 , train
acc :0.9620 , validation acc : 0.9609
Epoch [6/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9720 , validation acc : 0.9704
Epoch [7/100], Train Loss: 0.0001, Val Loss: 0.0004 , train
acc :0.9535 , validation acc : 0.9518
Epoch [8/100], Train Loss: 0.0001, Val Loss: 0.0001 , train
acc :0.9655 , validation acc : 0.9636
Epoch [9/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9783 , validation acc : 0.9766
Epoch [10/100], Train Loss: 0.0002, Val Loss: 0.0000 , train
acc :0.9554 , validation acc : 0.9543
Epoch [11/100], Train Loss: 0.0001, Val Loss: 0.0000 , train
acc :0.9785 , validation acc : 0.9764
Epoch [12/100], Train Loss: 0.0001, Val Loss: 0.0000 , train
acc :0.9816 , validation acc : 0.9793
Epoch [13/100], Train Loss: 0.0001, Val Loss: 0.0000 , train
acc :0.9750 , validation acc : 0.9724
Epoch [14/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9893 , validation acc : 0.9866
Epoch [15/100], Train Loss: 0.0005, Val Loss: 0.0011 , train
acc :0.8291 , validation acc : 0.8267
Epoch [16/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9920 , validation acc : 0.9892
Epoch [17/100], Train Loss: 0.0001, Val Loss: 0.0000 , train
acc :0.9735 , validation acc : 0.9711
Epoch [18/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9932 , validation acc : 0.9904
Epoch [19/100], Train Loss: 0.0001, Val Loss: 0.0000 , train
acc :0.9872 , validation acc : 0.9843
Epoch [20/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9907 , validation acc : 0.9875
Epoch [21/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9791 , validation acc : 0.9765
Epoch [22/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9942 , validation acc : 0.9914
Epoch [23/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9950 , validation acc : 0.9922
Epoch [24/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9968 , validation acc : 0.9936
Epoch [25/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9973 , validation acc : 0.9941
Epoch [26/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9962 , validation acc : 0.9926
Epoch [27/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9960 , validation acc : 0.9928
Epoch [28/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9985 , validation acc : 0.9950
Epoch [29/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
```

```
acc :0.9981 , validation acc : 0.9945
Epoch [30/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9989 , validation acc : 0.9954
Epoch [31/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9989 , validation acc : 0.9954
Stopping early!
training with learning rate = 0.05 gives accuracy= 0.9774

print(f"best test accuracy till now: {besttest_accuracy:.4f}")

# Dynamically set epochs based on the lengths of the loss/accuracy
lists
lengths = [
    len(train_loss1),
    len(train_loss2),
    len(train_loss3),
    len(train_loss4)
]

# Generate epochs for each case based on the specific list length
epochs1 = range(1, len(train_loss1) + 1)
epochs2 = range(1, len(train_loss2) + 1)
epochs3 = range(1, len(train_loss3) + 1)
epochs4 = range(1, len(train_loss4) + 1)

plt.figure(figsize=(12, 6))

# Loss plot
plt.subplot(1, 2, 1)
plt.plot(epochs1, train_loss1, label="Training Loss a=0.001")
plt.plot(epochs1, val_loss1, label="Validation Loss a=0.001")
plt.plot(epochs2, train_loss2, label="Training Loss a=0.0001")
plt.plot(epochs2, val_loss2, label="Validation Loss a=0.0001")
plt.plot(epochs3, train_loss3, label="Training Loss a=0.1")
plt.plot(epochs3, val_loss3, label="Validation Loss a=0.1")
plt.plot(epochs4, train_loss4, label="Training Loss a=0.05")
plt.plot(epochs4, val_loss4, label="Validation Loss a=0.05")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Training and Validation Loss")
plt.legend()

# Accuracy plot
plt.subplot(1, 2, 2)
plt.plot(epochs1, train_acc1, label="Training Accuracy a=0.001")
plt.plot(epochs1, val_acc1, label="Validation Accuracy a=0.001")
plt.plot(epochs2, train_acc2, label="Training Accuracy a=0.0001")
plt.plot(epochs2, val_acc2, label="Validation Accuracy a=0.0001")
plt.plot(epochs3, train_acc3, label="Training Accuracy a=0.1")
plt.plot(epochs3, val_acc3, label="Validation Accuracy a=0.1")
plt.plot(epochs4, train_acc4, label="Training Accuracy a=0.05")
plt.plot(epochs4, val_acc4, label="Validation Accuracy a=0.05")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Training and Validation Accuracy")
plt.legend()

plt.tight_layout()
plt.show()

best test accuracy till now: 0.9785
```



```
BATCH_SIZE=32
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE,
shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=BATCH_SIZE,
shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE,
shuffle=False)
model = FeedforwardNN()  # make a new neural network
criterion = nn.CrossEntropyLoss()  # Cross-entropy loss
optimizer = optim.SGD(model.parameters(), lr=0.01)
train_loss1,val_loss1,train_acc1,val_acc1=train_model(model,
train_loader, val_loader, criterion, optimizer, epochs=100)
correct = 0
total = 0
model.eval()  # Set the model to evaluation mode
with torch.no_grad():
    for images, labels in test_loader:
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)  # Get predicted class
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
test_accuracy=correct/total # till now this is the best test accuracy
print(test_accuracy)
if test_accuracy>besttest_accuracy:
  best_model=model
  besttest_accuracy=test_accuracy
```

```
Epoch [1/100], Train Loss: 0.0002, Val Loss: 0.0003 , train
acc :0.8948 , validation acc : 0.8945
Epoch [2/100], Train Loss: 0.0002, Val Loss: 0.0002 , train
acc :0.9057 , validation acc : 0.9052
Epoch [3/100], Train Loss: 0.0004, Val Loss: 0.0001 , train
acc :0.9270 , validation acc : 0.9264
Epoch [4/100], Train Loss: 0.0001, Val Loss: 0.0001 , train
acc :0.9389 , validation acc : 0.9384
Epoch [5/100], Train Loss: 0.0000, Val Loss: 0.0001 , train
acc :0.9456 , validation acc : 0.9448
Epoch [6/100], Train Loss: 0.0001, Val Loss: 0.0001 , train
acc :0.9499 , validation acc : 0.9489
Epoch [7/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9576 , validation acc : 0.9564
Epoch [8/100], Train Loss: 0.0001, Val Loss: 0.0000 , train
acc :0.9625 , validation acc : 0.9611
Epoch [9/100], Train Loss: 0.0003, Val Loss: 0.0001 , train
acc :0.9616 , validation acc : 0.9600
Epoch [10/100], Train Loss: 0.0001, Val Loss: 0.0000 , train
acc :0.9672 , validation acc : 0.9654
Epoch [11/100], Train Loss: 0.0001, Val Loss: 0.0000 , train
acc :0.9737 , validation acc : 0.9715
Epoch [12/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9730 , validation acc : 0.9710
Epoch [13/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9772 , validation acc : 0.9753
Epoch [14/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9762 , validation acc : 0.9741
Epoch [15/100], Train Loss: 0.0001, Val Loss: 0.0000 , train
acc :0.9783 , validation acc : 0.9761
Epoch [16/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9813 , validation acc : 0.9790
Epoch [17/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9833 , validation acc : 0.9810
Epoch [18/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9836 , validation acc : 0.9810
Epoch [19/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9837 , validation acc : 0.9814
Epoch [20/100], Train Loss: 0.0001, Val Loss: 0.0000 , train
acc :0.9859 , validation acc : 0.9832
Epoch [21/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
```

```
acc :0.9870 , validation acc : 0.9842
Epoch [22/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9863 , validation acc : 0.9838
Epoch [23/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9898 , validation acc : 0.9870
Epoch [24/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9909 , validation acc : 0.9881
Epoch [25/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9890 , validation acc : 0.9861
Epoch [26/100], Train Loss: 0.0001, Val Loss: 0.0000 , train
acc :0.9849 , validation acc : 0.9822
Epoch [27/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9913 , validation acc : 0.9885
Epoch [28/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9911 , validation acc : 0.9882
Epoch [29/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9925 , validation acc : 0.9895
Epoch [30/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9928 , validation acc : 0.9896
Epoch [31/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9946 , validation acc : 0.9916
Epoch [32/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9932 , validation acc : 0.9901
Epoch [33/100], Train Loss: 0.0001, Val Loss: 0.0000 , train
acc :0.9865 , validation acc : 0.9835
Epoch [34/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9947 , validation acc : 0.9915
Epoch [35/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9962 , validation acc : 0.9928
Epoch [36/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9949 , validation acc : 0.9915
Epoch [37/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9967 , validation acc : 0.9932
Epoch [38/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9963 , validation acc : 0.9929
Epoch [39/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9949 , validation acc : 0.9916
Epoch [40/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9975 , validation acc : 0.9939
Epoch [41/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9973 , validation acc : 0.9936
Epoch [42/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9967 , validation acc : 0.9928
Epoch [43/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9943 , validation acc : 0.9911
Stopping early!
0.9743
```

```
BATCH_SIZE=128
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE,
shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=BATCH_SIZE,
shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE,
shuffle=False)
model = FeedforwardNN()  # make a new neural network
criterion = nn.CrossEntropyLoss()  # Cross-entropy loss
optimizer = optim.SGD(model.parameters(), lr=0.01)
train_loss2,val_loss2,train_acc2,val_acc2=train_model(model,
train_loader, val_loader, criterion, optimizer, epochs=100)
correct = 0
total = 0
model.eval()  # Set the model to evaluation mode
with torch.no_grad():
    for images, labels in test_loader:
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)  # Get predicted class
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
test_accuracy=correct/total # till now this is the best test accuracy
print(test_accuracy)
if test_accuracy>besttest_accuracy:
  best_model=model
  besttest_accuracy=test_accuracy
```

```
Epoch [1/100], Train Loss: 0.0021, Val Loss: 0.0083 , train
acc :0.7896 , validation acc : 0.7899
Epoch [2/100], Train Loss: 0.0016, Val Loss: 0.0033 , train
acc :0.8665 , validation acc : 0.8654
Epoch [3/100], Train Loss: 0.0008, Val Loss: 0.0020 , train
acc :0.8845 , validation acc : 0.8839
Epoch [4/100], Train Loss: 0.0008, Val Loss: 0.0015 , train
acc :0.8963 , validation acc : 0.8959
Epoch [5/100], Train Loss: 0.0009, Val Loss: 0.0010 , train
acc :0.9036 , validation acc : 0.9034
Epoch [6/100], Train Loss: 0.0009, Val Loss: 0.0009 , train
acc :0.9093 , validation acc : 0.9092
Epoch [7/100], Train Loss: 0.0005, Val Loss: 0.0006 , train
acc :0.9113 , validation acc : 0.9108
Epoch [8/100], Train Loss: 0.0009, Val Loss: 0.0005 , train
acc :0.9126 , validation acc : 0.9120
Epoch [9/100], Train Loss: 0.0007, Val Loss: 0.0006 , train
acc :0.9196 , validation acc : 0.9191
Epoch [10/100], Train Loss: 0.0006, Val Loss: 0.0005 , train
acc :0.9225 , validation acc : 0.9218
Epoch [11/100], Train Loss: 0.0006, Val Loss: 0.0005 , train
acc :0.9246 , validation acc : 0.9238
Epoch [12/100], Train Loss: 0.0008, Val Loss: 0.0005 , train
acc :0.9270 , validation acc : 0.9264
Epoch [13/100], Train Loss: 0.0005, Val Loss: 0.0006 , train
```

```
acc :0.9308 , validation acc : 0.9306
Epoch [14/100], Train Loss: 0.0006, Val Loss: 0.0004 , train
acc :0.9323 , validation acc : 0.9316
Epoch [15/100], Train Loss: 0.0007, Val Loss: 0.0005 , train
acc :0.9355 , validation acc : 0.9349
Epoch [16/100], Train Loss: 0.0007, Val Loss: 0.0004 , train
acc :0.9379 , validation acc : 0.9373
Epoch [17/100], Train Loss: 0.0006, Val Loss: 0.0003 , train
acc :0.9384 , validation acc : 0.9375
Epoch [18/100], Train Loss: 0.0003, Val Loss: 0.0004 , train
acc :0.9411 , validation acc : 0.9403
Epoch [19/100], Train Loss: 0.0004, Val Loss: 0.0004 , train
acc :0.9432 , validation acc : 0.9424
Epoch [20/100], Train Loss: 0.0005, Val Loss: 0.0004 , train
acc :0.9463 , validation acc : 0.9454
Epoch [21/100], Train Loss: 0.0007, Val Loss: 0.0004 , train
acc :0.9465 , validation acc : 0.9457
Epoch [22/100], Train Loss: 0.0005, Val Loss: 0.0004 , train
acc :0.9485 , validation acc : 0.9475
Stopping early!
0.9455
```

```
BATCH_SIZE=512
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE,
shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=BATCH_SIZE,
shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE,
shuffle=False)
model = FeedforwardNN()  # make a new neural network
criterion = nn.CrossEntropyLoss()  # Cross-entropy loss
optimizer = optim.SGD(model.parameters(), lr=0.01)
train_loss3,val_loss3,train_acc3,val_acc3=train_model(model,
train_loader, val_loader, criterion, optimizer, epochs=100)
correct = 0
total = 0
model.eval()  # Set the model to evaluation mode
with torch.no_grad():
    for images, labels in test_loader:
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)  # Get predicted class
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
test_accuracy=correct/total # till now this is the best test accuracy
print(test_accuracy)
if test_accuracy>besttest_accuracy:
  best_model=model
  besttest_accuracy=test_accuracy
```

```
Epoch [1/100], Train Loss: 0.0215, Val Loss: 0.1097 , train
acc :0.3575 , validation acc : 0.3580
Epoch [2/100], Train Loss: 0.0174, Val Loss: 0.0885 , train
acc :0.5767 , validation acc : 0.5774
Epoch [3/100], Train Loss: 0.0126, Val Loss: 0.0631 , train
acc :0.7295 , validation acc : 0.7285
Epoch [4/100], Train Loss: 0.0085, Val Loss: 0.0463 , train
acc :0.7837 , validation acc : 0.7830
Epoch [5/100], Train Loss: 0.0071, Val Loss: 0.0376 , train
acc :0.8119 , validation acc : 0.8118
Epoch [6/100], Train Loss: 0.0070, Val Loss: 0.0328 , train
acc :0.8333 , validation acc : 0.8327
Epoch [7/100], Train Loss: 0.0054, Val Loss: 0.0298 , train
acc :0.8503 , validation acc : 0.8497
Epoch [8/100], Train Loss: 0.0044, Val Loss: 0.0279 , train
acc :0.8640 , validation acc : 0.8633
Epoch [9/100], Train Loss: 0.0043, Val Loss: 0.0259 , train
acc :0.8727 , validation acc : 0.8718
Epoch [10/100], Train Loss: 0.0052, Val Loss: 0.0247 , train
acc :0.8791 , validation acc : 0.8787
Epoch [11/100], Train Loss: 0.0041, Val Loss: 0.0236 , train
acc :0.8842 , validation acc : 0.8839
Epoch [12/100], Train Loss: 0.0037, Val Loss: 0.0226 , train
acc :0.8884 , validation acc : 0.8881
Epoch [13/100], Train Loss: 0.0034, Val Loss: 0.0220 , train
acc :0.8916 , validation acc : 0.8911
Epoch [14/100], Train Loss: 0.0049, Val Loss: 0.0212 , train
acc :0.8948 , validation acc : 0.8945
Epoch [15/100], Train Loss: 0.0037, Val Loss: 0.0205 , train
acc :0.8964 , validation acc : 0.8960
Epoch [16/100], Train Loss: 0.0033, Val Loss: 0.0203 , train
acc :0.8993 , validation acc : 0.8992
Epoch [17/100], Train Loss: 0.0044, Val Loss: 0.0199 , train
acc :0.9006 , validation acc : 0.9006
Epoch [18/100], Train Loss: 0.0033, Val Loss: 0.0194 , train
acc :0.9027 , validation acc : 0.9027
Epoch [19/100], Train Loss: 0.0037, Val Loss: 0.0190 , train
acc :0.9045 , validation acc : 0.9045
Epoch [20/100], Train Loss: 0.0038, Val Loss: 0.0187 , train
acc :0.9063 , validation acc : 0.9061
Epoch [21/100], Train Loss: 0.0026, Val Loss: 0.0185 , train
acc :0.9066 , validation acc : 0.9067
Epoch [22/100], Train Loss: 0.0028, Val Loss: 0.0182 , train
acc :0.9087 , validation acc : 0.9085
Epoch [23/100], Train Loss: 0.0029, Val Loss: 0.0179 , train
acc :0.9086 , validation acc : 0.9084
Epoch [24/100], Train Loss: 0.0031, Val Loss: 0.0179 , train
acc :0.9108 , validation acc : 0.9108
Epoch [25/100], Train Loss: 0.0031, Val Loss: 0.0175 , train
acc :0.9111 , validation acc : 0.9111
```

```
Epoch [26/100], Train Loss: 0.0030, Val Loss: 0.0174 , train
acc :0.9128 , validation acc : 0.9126
Epoch [27/100], Train Loss: 0.0022, Val Loss: 0.0171 , train
acc :0.9136 , validation acc : 0.9136
Epoch [28/100], Train Loss: 0.0027, Val Loss: 0.0172 , train
acc :0.9141 , validation acc : 0.9138
Epoch [29/100], Train Loss: 0.0027, Val Loss: 0.0167 , train
acc :0.9156 , validation acc : 0.9153
Epoch [30/100], Train Loss: 0.0027, Val Loss: 0.0166 , train
acc :0.9158 , validation acc : 0.9157
Epoch [31/100], Train Loss: 0.0028, Val Loss: 0.0164 , train
acc :0.9172 , validation acc : 0.9169
Epoch [32/100], Train Loss: 0.0025, Val Loss: 0.0164 , train
acc :0.9174 , validation acc : 0.9173
Epoch [33/100], Train Loss: 0.0039, Val Loss: 0.0163 , train
acc :0.9188 , validation acc : 0.9186
Epoch [34/100], Train Loss: 0.0029, Val Loss: 0.0160 , train
acc :0.9191 , validation acc : 0.9188
Epoch [35/100], Train Loss: 0.0024, Val Loss: 0.0160 , train
acc :0.9206 , validation acc : 0.9205
Epoch [36/100], Train Loss: 0.0033, Val Loss: 0.0159 , train
acc :0.9214 , validation acc : 0.9214
Epoch [37/100], Train Loss: 0.0025, Val Loss: 0.0158 , train
acc :0.9210 , validation acc : 0.9204
Epoch [38/100], Train Loss: 0.0025, Val Loss: 0.0156 , train
acc :0.9218 , validation acc : 0.9214
Epoch [39/100], Train Loss: 0.0029, Val Loss: 0.0155 , train
acc :0.9234 , validation acc : 0.9233
Epoch [40/100], Train Loss: 0.0031, Val Loss: 0.0154 , train
acc :0.9230 , validation acc : 0.9229
Epoch [41/100], Train Loss: 0.0027, Val Loss: 0.0154 , train
acc :0.9241 , validation acc : 0.9239
Epoch [42/100], Train Loss: 0.0029, Val Loss: 0.0152 , train
acc :0.9243 , validation acc : 0.9240
Epoch [43/100], Train Loss: 0.0027, Val Loss: 0.0151 , train
acc :0.9257 , validation acc : 0.9253
Epoch [44/100], Train Loss: 0.0022, Val Loss: 0.0150 , train
acc :0.9260 , validation acc : 0.9255
Epoch [45/100], Train Loss: 0.0024, Val Loss: 0.0148 , train
acc :0.9267 , validation acc : 0.9263
Epoch [46/100], Train Loss: 0.0020, Val Loss: 0.0148 , train
acc :0.9265 , validation acc : 0.9262
Epoch [47/100], Train Loss: 0.0032, Val Loss: 0.0146 , train
acc :0.9278 , validation acc : 0.9272
Epoch [48/100], Train Loss: 0.0028, Val Loss: 0.0148 , train
acc :0.9266 , validation acc : 0.9263
Epoch [49/100], Train Loss: 0.0026, Val Loss: 0.0146 , train
acc :0.9292 , validation acc : 0.9287
Epoch [50/100], Train Loss: 0.0024, Val Loss: 0.0144 , train
acc :0.9288 , validation acc : 0.9284
```

```
Epoch [51/100], Train Loss: 0.0022, Val Loss: 0.0143 , train
acc :0.9299 , validation acc : 0.9296
Epoch [52/100], Train Loss: 0.0023, Val Loss: 0.0142 , train
acc :0.9302 , validation acc : 0.9297
Epoch [53/100], Train Loss: 0.0031, Val Loss: 0.0142 , train
acc :0.9312 , validation acc : 0.9306
Epoch [54/100], Train Loss: 0.0027, Val Loss: 0.0141 , train
acc :0.9315 , validation acc : 0.9308
Epoch [55/100], Train Loss: 0.0030, Val Loss: 0.0140 , train
acc :0.9330 , validation acc : 0.9323
Epoch [56/100], Train Loss: 0.0027, Val Loss: 0.0138 , train
acc :0.9328 , validation acc : 0.9321
Epoch [57/100], Train Loss: 0.0019, Val Loss: 0.0138 , train
acc :0.9326 , validation acc : 0.9318
Epoch [58/100], Train Loss: 0.0021, Val Loss: 0.0135 , train
acc :0.9334 , validation acc : 0.9329
Epoch [59/100], Train Loss: 0.0023, Val Loss: 0.0135 , train
acc :0.9344 , validation acc : 0.9337
Epoch [60/100], Train Loss: 0.0012, Val Loss: 0.0134 , train
acc :0.9350 , validation acc : 0.9344
Epoch [61/100], Train Loss: 0.0025, Val Loss: 0.0132 , train
acc :0.9355 , validation acc : 0.9348
Epoch [62/100], Train Loss: 0.0021, Val Loss: 0.0134 , train
acc :0.9362 , validation acc : 0.9352
Epoch [63/100], Train Loss: 0.0022, Val Loss: 0.0133 , train
acc :0.9355 , validation acc : 0.9347
Epoch [64/100], Train Loss: 0.0021, Val Loss: 0.0130 , train
acc :0.9375 , validation acc : 0.9367
Epoch [65/100], Train Loss: 0.0022, Val Loss: 0.0132 , train
acc :0.9374 , validation acc : 0.9366
Epoch [66/100], Train Loss: 0.0023, Val Loss: 0.0128 , train
acc :0.9381 , validation acc : 0.9373
Epoch [67/100], Train Loss: 0.0029, Val Loss: 0.0129 , train
acc :0.9382 , validation acc : 0.9373
Epoch [68/100], Train Loss: 0.0019, Val Loss: 0.0128 , train
acc :0.9386 , validation acc : 0.9379
Epoch [69/100], Train Loss: 0.0017, Val Loss: 0.0126 , train
acc :0.9392 , validation acc : 0.9382
Epoch [70/100], Train Loss: 0.0020, Val Loss: 0.0129 , train
acc :0.9388 , validation acc : 0.9377
Epoch [71/100], Train Loss: 0.0021, Val Loss: 0.0125 , train
acc :0.9409 , validation acc : 0.9398
Epoch [72/100], Train Loss: 0.0029, Val Loss: 0.0124 , train
acc :0.9405 , validation acc : 0.9396
Epoch [73/100], Train Loss: 0.0027, Val Loss: 0.0124 , train
acc :0.9413 , validation acc : 0.9404
Epoch [74/100], Train Loss: 0.0015, Val Loss: 0.0123 , train
acc :0.9424 , validation acc : 0.9413
Epoch [75/100], Train Loss: 0.0021, Val Loss: 0.0122 , train
acc :0.9423 , validation acc : 0.9413
```

```
Epoch [76/100], Train Loss: 0.0021, Val Loss: 0.0120 , train
acc :0.9433 , validation acc : 0.9422
Epoch [77/100], Train Loss: 0.0022, Val Loss: 0.0120 , train
acc :0.9434 , validation acc : 0.9423
Epoch [78/100], Train Loss: 0.0016, Val Loss: 0.0119 , train
acc :0.9443 , validation acc : 0.9431
Epoch [79/100], Train Loss: 0.0025, Val Loss: 0.0118 , train
acc :0.9446 , validation acc : 0.9434
Epoch [80/100], Train Loss: 0.0027, Val Loss: 0.0119 , train
acc :0.9447 , validation acc : 0.9436
Epoch [81/100], Train Loss: 0.0025, Val Loss: 0.0117 , train
acc :0.9443 , validation acc : 0.9433
Epoch [82/100], Train Loss: 0.0028, Val Loss: 0.0117 , train
acc :0.9459 , validation acc : 0.9448
Epoch [83/100], Train Loss: 0.0015, Val Loss: 0.0117 , train
acc :0.9465 , validation acc : 0.9453
Epoch [84/100], Train Loss: 0.0017, Val Loss: 0.0116 , train
acc :0.9470 , validation acc : 0.9458
Epoch [85/100], Train Loss: 0.0023, Val Loss: 0.0115 , train
acc :0.9464 , validation acc : 0.9453
Epoch [86/100], Train Loss: 0.0016, Val Loss: 0.0114 , train
acc :0.9477 , validation acc : 0.9464
Epoch [87/100], Train Loss: 0.0020, Val Loss: 0.0112 , train
acc :0.9480 , validation acc : 0.9468
Epoch [88/100], Train Loss: 0.0019, Val Loss: 0.0112 , train
acc :0.9483 , validation acc : 0.9472
Epoch [89/100], Train Loss: 0.0020, Val Loss: 0.0113 , train
acc :0.9483 , validation acc : 0.9474
Epoch [90/100], Train Loss: 0.0016, Val Loss: 0.0111 , train
acc :0.9490 , validation acc : 0.9478
Epoch [91/100], Train Loss: 0.0016, Val Loss: 0.0112 , train
acc :0.9500 , validation acc : 0.9487
Epoch [92/100], Train Loss: 0.0019, Val Loss: 0.0111 , train
acc :0.9496 , validation acc : 0.9484
Epoch [93/100], Train Loss: 0.0017, Val Loss: 0.0108 , train
acc :0.9504 , validation acc : 0.9492
Epoch [94/100], Train Loss: 0.0012, Val Loss: 0.0107 , train
acc :0.9510 , validation acc : 0.9498
Epoch [95/100], Train Loss: 0.0022, Val Loss: 0.0107 , train
acc :0.9508 , validation acc : 0.9496
Epoch [96/100], Train Loss: 0.0025, Val Loss: 0.0106 , train
acc :0.9516 , validation acc : 0.9502
Epoch [97/100], Train Loss: 0.0014, Val Loss: 0.0108 , train
acc :0.9514 , validation acc : 0.9501
Epoch [98/100], Train Loss: 0.0021, Val Loss: 0.0105 , train
acc :0.9530 , validation acc : 0.9515
Epoch [99/100], Train Loss: 0.0019, Val Loss: 0.0106 , train
acc :0.9531 , validation acc : 0.9515
Epoch [100/100], Train Loss: 0.0018, Val Loss: 0.0105 , train
```

```
acc :0.9528 , validation acc : 0.9514
0.9488

BATCH_SIZE=2048
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE,
shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=BATCH_SIZE,
shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE,
shuffle=False)
model = FeedforwardNN()  # make a new neural network
criterion = nn.CrossEntropyLoss()  # Cross-entropy loss
optimizer = optim.SGD(model.parameters(), lr=0.01)
train_loss4,val_loss4,train_acc4,val_acc4=train_model(model,
train_loader, val_loader, criterion, optimizer, epochs=100)
correct = 0
total = 0
model.eval()  # Set the model to evaluation mode
with torch.no_grad():
    for images, labels in test_loader:
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)  # Get predicted class
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
test_accuracy=correct/total # till now this is the best test accuracy
print(test_accuracy)
if test_accuracy>besttest_accuracy:
  best_model=model
  besttest_accuracy=test_accuracy

Epoch [1/100], Train Loss: 0.0942, Val Loss: 0.5655 , train
acc :0.1944 , validation acc : 0.1942
Epoch [2/100], Train Loss: 0.0925, Val Loss: 0.5548 , train
acc :0.2760 , validation acc : 0.2759
Epoch [3/100], Train Loss: 0.0906, Val Loss: 0.5429 , train
acc :0.3559 , validation acc : 0.3558
Epoch [4/100], Train Loss: 0.0882, Val Loss: 0.5282 , train
acc :0.4215 , validation acc : 0.4213
Epoch [5/100], Train Loss: 0.0858, Val Loss: 0.5098 , train
acc :0.4743 , validation acc : 0.4740
Epoch [6/100], Train Loss: 0.0819, Val Loss: 0.4870 , train
acc :0.5193 , validation acc : 0.5192
Epoch [7/100], Train Loss: 0.0762, Val Loss: 0.4592 , train
acc :0.5599 , validation acc : 0.5601
Epoch [8/100], Train Loss: 0.0706, Val Loss: 0.4270 , train
acc :0.6131 , validation acc : 0.6133
Epoch [9/100], Train Loss: 0.0651, Val Loss: 0.3920 , train
acc :0.6651 , validation acc : 0.6655
Epoch [10/100], Train Loss: 0.0603, Val Loss: 0.3570 , train
acc :0.7026 , validation acc : 0.7027
```

```
Epoch [11/100], Train Loss: 0.0537, Val Loss: 0.3238 , train
acc :0.7305 , validation acc : 0.7303
Epoch [12/100], Train Loss: 0.0489, Val Loss: 0.2940 , train
acc :0.7554 , validation acc : 0.7554
Epoch [13/100], Train Loss: 0.0449, Val Loss: 0.2679 , train
acc :0.7796 , validation acc : 0.7789
Epoch [14/100], Train Loss: 0.0424, Val Loss: 0.2454 , train
acc :0.7941 , validation acc : 0.7942
Epoch [15/100], Train Loss: 0.0366, Val Loss: 0.2266 , train
acc :0.8017 , validation acc : 0.8013
Epoch [16/100], Train Loss: 0.0363, Val Loss: 0.2105 , train
acc :0.8100 , validation acc : 0.8100
Epoch [17/100], Train Loss: 0.0325, Val Loss: 0.1970 , train
acc :0.8186 , validation acc : 0.8184
Epoch [18/100], Train Loss: 0.0311, Val Loss: 0.1857 , train
acc :0.8254 , validation acc : 0.8253
Epoch [19/100], Train Loss: 0.0282, Val Loss: 0.1759 , train
acc :0.8306 , validation acc : 0.8306
Epoch [20/100], Train Loss: 0.0283, Val Loss: 0.1675 , train
acc :0.8358 , validation acc : 0.8357
Epoch [21/100], Train Loss: 0.0261, Val Loss: 0.1601 , train
acc :0.8398 , validation acc : 0.8400
Epoch [22/100], Train Loss: 0.0248, Val Loss: 0.1540 , train
acc :0.8447 , validation acc : 0.8447
Epoch [23/100], Train Loss: 0.0241, Val Loss: 0.1483 , train
acc :0.8482 , validation acc : 0.8484
Epoch [24/100], Train Loss: 0.0244, Val Loss: 0.1433 , train
acc :0.8522 , validation acc : 0.8522
Epoch [25/100], Train Loss: 0.0216, Val Loss: 0.1391 , train
acc :0.8551 , validation acc : 0.8552
Epoch [26/100], Train Loss: 0.0219, Val Loss: 0.1349 , train
acc :0.8577 , validation acc : 0.8577
Epoch [27/100], Train Loss: 0.0243, Val Loss: 0.1312 , train
acc :0.8625 , validation acc : 0.8624
Epoch [28/100], Train Loss: 0.0209, Val Loss: 0.1280 , train
acc :0.8644 , validation acc : 0.8644
Epoch [29/100], Train Loss: 0.0209, Val Loss: 0.1250 , train
acc :0.8670 , validation acc : 0.8670
Epoch [30/100], Train Loss: 0.0198, Val Loss: 0.1223 , train
acc :0.8684 , validation acc : 0.8683
Epoch [31/100], Train Loss: 0.0194, Val Loss: 0.1200 , train
acc :0.8705 , validation acc : 0.8702
Epoch [32/100], Train Loss: 0.0195, Val Loss: 0.1179 , train
acc :0.8717 , validation acc : 0.8716
Epoch [33/100], Train Loss: 0.0205, Val Loss: 0.1156 , train
acc :0.8754 , validation acc : 0.8754
Epoch [34/100], Train Loss: 0.0198, Val Loss: 0.1136 , train
acc :0.8764 , validation acc : 0.8760
Epoch [35/100], Train Loss: 0.0185, Val Loss: 0.1118 , train
```

```
acc :0.8785 , validation acc : 0.8782
Epoch [36/100], Train Loss: 0.0166, Val Loss: 0.1102 , train
acc :0.8793 , validation acc : 0.8791
Epoch [37/100], Train Loss: 0.0169, Val Loss: 0.1087 , train
acc :0.8801 , validation acc : 0.8797
Epoch [38/100], Train Loss: 0.0170, Val Loss: 0.1070 , train
acc :0.8824 , validation acc : 0.8821
Epoch [39/100], Train Loss: 0.0177, Val Loss: 0.1057 , train
acc :0.8835 , validation acc : 0.8831
Epoch [40/100], Train Loss: 0.0171, Val Loss: 0.1047 , train
acc :0.8827 , validation acc : 0.8824
Epoch [41/100], Train Loss: 0.0153, Val Loss: 0.1033 , train
acc :0.8853 , validation acc : 0.8848
Epoch [42/100], Train Loss: 0.0173, Val Loss: 0.1022 , train
acc :0.8859 , validation acc : 0.8857
Epoch [43/100], Train Loss: 0.0140, Val Loss: 0.1012 , train
acc :0.8873 , validation acc : 0.8869
Epoch [44/100], Train Loss: 0.0152, Val Loss: 0.1002 , train
acc :0.8880 , validation acc : 0.8879
Epoch [45/100], Train Loss: 0.0172, Val Loss: 0.0995 , train
acc :0.8889 , validation acc : 0.8885
Epoch [46/100], Train Loss: 0.0170, Val Loss: 0.0985 , train
acc :0.8895 , validation acc : 0.8893
Epoch [47/100], Train Loss: 0.0183, Val Loss: 0.0980 , train
acc :0.8893 , validation acc : 0.8891
Epoch [48/100], Train Loss: 0.0185, Val Loss: 0.0967 , train
acc :0.8911 , validation acc : 0.8907
Epoch [49/100], Train Loss: 0.0165, Val Loss: 0.0958 , train
acc :0.8925 , validation acc : 0.8921
Epoch [50/100], Train Loss: 0.0150, Val Loss: 0.0953 , train
acc :0.8925 , validation acc : 0.8923
Epoch [51/100], Train Loss: 0.0164, Val Loss: 0.0945 , train
acc :0.8936 , validation acc : 0.8934
Epoch [52/100], Train Loss: 0.0158, Val Loss: 0.0940 , train
acc :0.8937 , validation acc : 0.8933
Epoch [53/100], Train Loss: 0.0164, Val Loss: 0.0932 , train
acc :0.8945 , validation acc : 0.8942
Epoch [54/100], Train Loss: 0.0136, Val Loss: 0.0929 , train
acc :0.8945 , validation acc : 0.8942
Epoch [55/100], Train Loss: 0.0140, Val Loss: 0.0920 , train
acc :0.8960 , validation acc : 0.8958
Epoch [56/100], Train Loss: 0.0181, Val Loss: 0.0915 , train
acc :0.8967 , validation acc : 0.8965
Epoch [57/100], Train Loss: 0.0139, Val Loss: 0.0909 , train
acc :0.8969 , validation acc : 0.8968
Epoch [58/100], Train Loss: 0.0153, Val Loss: 0.0904 , train
acc :0.8969 , validation acc : 0.8967
Epoch [59/100], Train Loss: 0.0168, Val Loss: 0.0901 , train
acc :0.8979 , validation acc : 0.8974
```

```
Epoch [60/100], Train Loss: 0.0148, Val Loss: 0.0891 , train
acc :0.8988 , validation acc : 0.8984
Epoch [61/100], Train Loss: 0.0166, Val Loss: 0.0893 , train
acc :0.8988 , validation acc : 0.8984
Epoch [62/100], Train Loss: 0.0136, Val Loss: 0.0883 , train
acc :0.8998 , validation acc : 0.8995
Epoch [63/100], Train Loss: 0.0154, Val Loss: 0.0880 , train
acc :0.8995 , validation acc : 0.8992
Epoch [64/100], Train Loss: 0.0141, Val Loss: 0.0874 , train
acc :0.9009 , validation acc : 0.9006
Epoch [65/100], Train Loss: 0.0127, Val Loss: 0.0872 , train
acc :0.9012 , validation acc : 0.9010
Epoch [66/100], Train Loss: 0.0139, Val Loss: 0.0870 , train
acc :0.9009 , validation acc : 0.9007
Epoch [67/100], Train Loss: 0.0163, Val Loss: 0.0866 , train
acc :0.9016 , validation acc : 0.9012
Epoch [68/100], Train Loss: 0.0155, Val Loss: 0.0863 , train
acc :0.9019 , validation acc : 0.9017
Epoch [69/100], Train Loss: 0.0143, Val Loss: 0.0860 , train
acc :0.9022 , validation acc : 0.9018
Epoch [70/100], Train Loss: 0.0133, Val Loss: 0.0853 , train
acc :0.9027 , validation acc : 0.9024
Epoch [71/100], Train Loss: 0.0143, Val Loss: 0.0853 , train
acc :0.9029 , validation acc : 0.9027
Epoch [72/100], Train Loss: 0.0147, Val Loss: 0.0847 , train
acc :0.9035 , validation acc : 0.9032
Epoch [73/100], Train Loss: 0.0131, Val Loss: 0.0843 , train
acc :0.9036 , validation acc : 0.9033
Epoch [74/100], Train Loss: 0.0140, Val Loss: 0.0839 , train
acc :0.9040 , validation acc : 0.9039
Epoch [75/100], Train Loss: 0.0127, Val Loss: 0.0838 , train
acc :0.9047 , validation acc : 0.9045
Epoch [76/100], Train Loss: 0.0124, Val Loss: 0.0835 , train
acc :0.9051 , validation acc : 0.9048
Epoch [77/100], Train Loss: 0.0141, Val Loss: 0.0831 , train
acc :0.9054 , validation acc : 0.9053
Epoch [78/100], Train Loss: 0.0132, Val Loss: 0.0828 , train
acc :0.9061 , validation acc : 0.9059
Epoch [79/100], Train Loss: 0.0126, Val Loss: 0.0828 , train
acc :0.9056 , validation acc : 0.9053
Epoch [80/100], Train Loss: 0.0126, Val Loss: 0.0826 , train
acc :0.9060 , validation acc : 0.9058
Epoch [81/100], Train Loss: 0.0126, Val Loss: 0.0822 , train
acc :0.9064 , validation acc : 0.9062
Epoch [82/100], Train Loss: 0.0140, Val Loss: 0.0818 , train
acc :0.9064 , validation acc : 0.9062
Epoch [83/100], Train Loss: 0.0134, Val Loss: 0.0819 , train
acc :0.9063 , validation acc : 0.9062
Epoch [84/100], Train Loss: 0.0120, Val Loss: 0.0815 , train
```

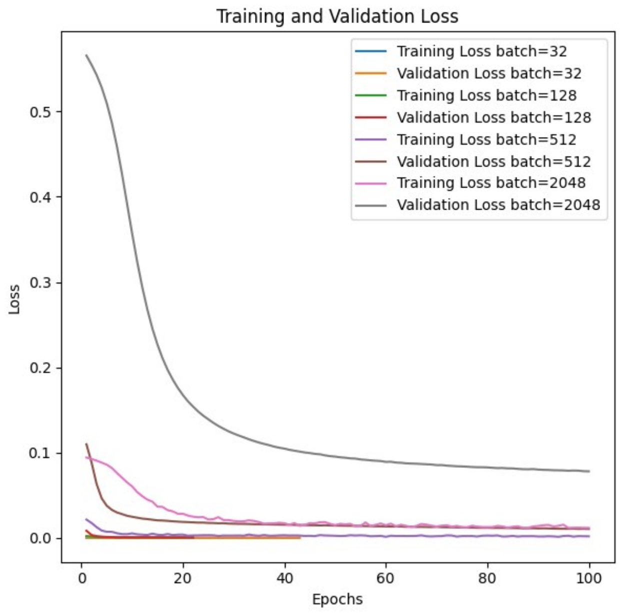
```
acc :0.9071 , validation acc : 0.9068
Epoch [85/100], Train Loss: 0.0130, Val Loss: 0.0814 , train
acc :0.9072 , validation acc : 0.9070
Epoch [86/100], Train Loss: 0.0135, Val Loss: 0.0809 , train
acc :0.9082 , validation acc : 0.9080
Epoch [87/100], Train Loss: 0.0119, Val Loss: 0.0805 , train
acc :0.9082 , validation acc : 0.9080
Epoch [88/100], Train Loss: 0.0118, Val Loss: 0.0804 , train
acc :0.9080 , validation acc : 0.9077
Epoch [89/100], Train Loss: 0.0133, Val Loss: 0.0806 , train
acc :0.9084 , validation acc : 0.9081
Epoch [90/100], Train Loss: 0.0139, Val Loss: 0.0801 , train
acc :0.9091 , validation acc : 0.9089
Epoch [91/100], Train Loss: 0.0147, Val Loss: 0.0798 , train
acc :0.9095 , validation acc : 0.9093
Epoch [92/100], Train Loss: 0.0152, Val Loss: 0.0796 , train
acc :0.9097 , validation acc : 0.9095
Epoch [93/100], Train Loss: 0.0134, Val Loss: 0.0793 , train
acc :0.9101 , validation acc : 0.9100
Epoch [94/100], Train Loss: 0.0140, Val Loss: 0.0792 , train
acc :0.9104 , validation acc : 0.9102
Epoch [95/100], Train Loss: 0.0154, Val Loss: 0.0791 , train
acc :0.9106 , validation acc : 0.9103
Epoch [96/100], Train Loss: 0.0116, Val Loss: 0.0787 , train
acc :0.9105 , validation acc : 0.9103
Epoch [97/100], Train Loss: 0.0119, Val Loss: 0.0789 , train
acc :0.9107 , validation acc : 0.9105
Epoch [98/100], Train Loss: 0.0120, Val Loss: 0.0788 , train
acc :0.9106 , validation acc : 0.9102
Epoch [99/100], Train Loss: 0.0117, Val Loss: 0.0782 , train
acc :0.9115 , validation acc : 0.9111
Epoch [100/100], Train Loss: 0.0117, Val Loss: 0.0780 , train
acc :0.9110 , validation acc : 0.9108
0.914
```

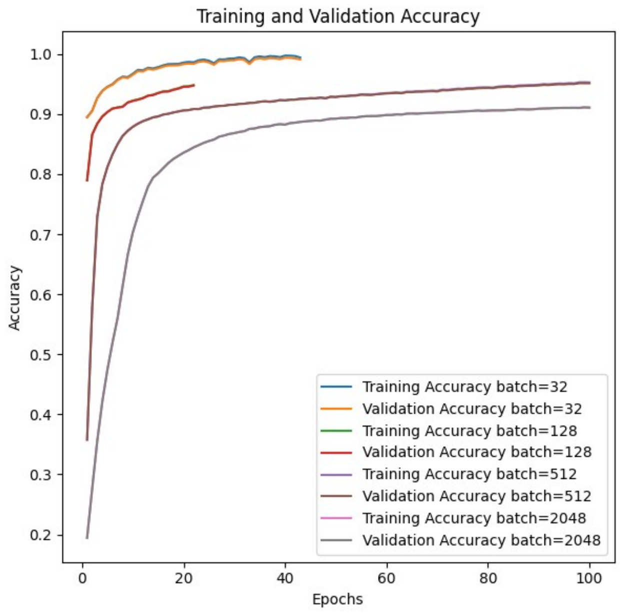
How Batch Size Affects Model Performance
Small Batch Sizes:

Training Speed: Slower due to more weight updates.
Model Accuracy: Can lead to better generalization but noisier convergence.
Memory Usage: Efficient on GPUs with limited memory.
Large Batch Sizes:
Training Speed: Faster per epoch, but total time to convergence may increase.
Model Accuracy: May result in poorer generalization if the batch is too large.
Memory Usage: High memory consumption, requiring powerful GPUs.

```
print(besttest_accuracy)

# Dynamically set epochs based on the lengths of the loss/accuracy
```

```
lists
lengths = [
    len(train_loss1),
    len(train_loss2),
    len(train_loss3),
    len(train_loss4)
]

# Generate epochs for each case based on the specific list length
epochs1 = range(1, len(train_loss1) + 1)
epochs2 = range(1, len(train_loss2) + 1)
epochs3 = range(1, len(train_loss3) + 1)
epochs4 = range(1, len(train_loss4) + 1)

plt.figure(figsize=(12, 6))

# Loss plot
plt.subplot(1, 2, 1)
plt.plot(epochs1, train_loss1, label="Training Loss batch=32")
plt.plot(epochs1, val_loss1, label="Validation Loss batch=32")
plt.plot(epochs2, train_loss2, label="Training Loss batch=128")
plt.plot(epochs2, val_loss2, label="Validation Loss batch=128")
plt.plot(epochs3, train_loss3, label="Training Loss batch=512")
plt.plot(epochs3, val_loss3, label="Validation Loss batch=512")
plt.plot(epochs4, train_loss4, label="Training Loss batch=2048")
plt.plot(epochs4, val_loss4, label="Validation Loss batch=2048")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Training and Validation Loss")
plt.legend()

# Accuracy plot
plt.subplot(1, 2, 2)
plt.plot(epochs1, train_acc1, label="Training Accuracy batch=32")
plt.plot(epochs1, val_acc1, label="Validation Accuracy batch=32")
plt.plot(epochs2, train_acc2, label="Training Accuracy batch=128")
plt.plot(epochs2, val_acc2, label="Validation Accuracy batch=128")
plt.plot(epochs3, train_acc3, label="Training Accuracy batch=512")
plt.plot(epochs3, val_acc3, label="Validation Accuracy batch=512")
plt.plot(epochs4, train_acc4, label="Training Accuracy batch=2048")
plt.plot(epochs4, val_acc4, label="Validation Accuracy batch=2048")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Training and Validation Accuracy")
plt.legend()

plt.tight_layout()
plt.show()

0.9785
```





```
BATCH_SIZE=64
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE,
shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=BATCH_SIZE,
shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE,
shuffle=False)

# Define the feedforward neural network
class FeedforwardNN2(nn.Module):  #neural network defined
    def __init__(self, input_size=784, hidden_size1=128,
hidden_size2=64,hidden_size3=32, output_size=10):
        super(FeedforwardNN2, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size1) # make the
y=w^T x + b
        self.relu1 = nn.ReLU()  # make activation function 1
        self.fc2 = nn.Linear(hidden_size1, hidden_size2)
        self.relu2 = nn.ReLU()  # make activation function 2
        self.fc3 = nn.Linear(hidden_size2, hidden_size3)
        self.relu3 = nn.ReLU()  # make activation function 2
        self.fc4 = nn.Linear(hidden_size3, output_size)  # applying
linearity for output
    def forward(self, x):
        x = x.view(x.size(0), -1)  # Flatten the input  make it 1-d
        x = self.fc1(x)  # Apply the first linear layer
        x = self.relu1(x)  # Apply the activation function
        x = self.fc2(x)  # Apply the second linear layer
        x = self.relu2(x)  # Apply the activation function
        x = self.fc3(x)  # Apply the output linear layer
        x = self.relu3(x)  # Apply the activation function
        x = self.fc4(x)  # Apply the output linear layer
        return x  # return the output
# Initialize the model, loss function, and optimizer
model = FeedforwardNN()  # make a new neural network
criterion = nn.CrossEntropyLoss()  # Cross-entropy loss
optimizer = optim.SGD(model.parameters(), lr=0.01)  # Gradient
Stochastic Descent

model1 = FeedforwardNN(hidden_size1=64, hidden_size2=32)  # make a new
neural network
criterion = nn.CrossEntropyLoss()  # Cross-entropy loss
optimizer = optim.SGD(model1.parameters(), lr=0.01)
train_loss1,val_loss1,train_acc1,val_acc1=train_model(model1,
train_loader, val_loader, criterion, optimizer, epochs=100)
correct = 0
total = 0
model1.eval()  # Set the model to evaluation mode
with torch.no_grad():
    for images, labels in test_loader:
        outputs = model1(images)
        _, predicted = torch.max(outputs, 1)  # Get predicted class
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
test_accuracy=correct/total # till now this is the best test accuracy
print(test_accuracy)
if test_accuracy>besttest_accuracy:
  best_model=model1
  besttest_accuracy=test_accuracy


Epoch [1/100], Train Loss: 0.0005, Val Loss: 0.0021 , train
acc :0.8363 , validation acc : 0.8352
Epoch [2/100], Train Loss: 0.0004, Val Loss: 0.0010 , train
acc :0.8842 , validation acc : 0.8835
Epoch [3/100], Train Loss: 0.0001, Val Loss: 0.0004 , train
acc :0.9019 , validation acc : 0.9014
Epoch [4/100], Train Loss: 0.0002, Val Loss: 0.0004 , train
acc :0.9106 , validation acc : 0.9101
Epoch [5/100], Train Loss: 0.0001, Val Loss: 0.0003 , train
acc :0.9168 , validation acc : 0.9163
Epoch [6/100], Train Loss: 0.0001, Val Loss: 0.0005 , train
acc :0.9152 , validation acc : 0.9145
Epoch [7/100], Train Loss: 0.0002, Val Loss: 0.0003 , train
acc :0.9218 , validation acc : 0.9214
Epoch [8/100], Train Loss: 0.0002, Val Loss: 0.0002 , train
acc :0.9281 , validation acc : 0.9277
Epoch [9/100], Train Loss: 0.0002, Val Loss: 0.0005 , train
acc :0.9310 , validation acc : 0.9306
Epoch [10/100], Train Loss: 0.0002, Val Loss: 0.0002 , train
acc :0.9374 , validation acc : 0.9369
Epoch [11/100], Train Loss: 0.0005, Val Loss: 0.0002 , train
```

```
acc :0.9368 , validation acc : 0.9357
Epoch [12/100], Train Loss: 0.0001, Val Loss: 0.0003 , train
acc :0.9479 , validation acc : 0.9471
Epoch [13/100], Train Loss: 0.0001, Val Loss: 0.0002 , train
acc :0.9484 , validation acc : 0.9476
Epoch [14/100], Train Loss: 0.0005, Val Loss: 0.0003 , train
acc :0.9347 , validation acc : 0.9331
Epoch [15/100], Train Loss: 0.0002, Val Loss: 0.0002 , train
acc :0.9494 , validation acc : 0.9482
Epoch [16/100], Train Loss: 0.0001, Val Loss: 0.0002 , train
acc :0.9527 , validation acc : 0.9517
Epoch [17/100], Train Loss: 0.0000, Val Loss: 0.0001 , train
acc :0.9577 , validation acc : 0.9562
Epoch [18/100], Train Loss: 0.0001, Val Loss: 0.0001 , train
acc :0.9586 , validation acc : 0.9568
Epoch [19/100], Train Loss: 0.0001, Val Loss: 0.0001 , train
acc :0.9619 , validation acc : 0.9601
Epoch [20/100], Train Loss: 0.0005, Val Loss: 0.0001 , train
acc :0.9498 , validation acc : 0.9483
Epoch [21/100], Train Loss: 0.0002, Val Loss: 0.0001 , train
acc :0.9530 , validation acc : 0.9518
Epoch [22/100], Train Loss: 0.0000, Val Loss: 0.0001 , train
acc :0.9648 , validation acc : 0.9629
Epoch [23/100], Train Loss: 0.0001, Val Loss: 0.0001 , train
acc :0.9670 , validation acc : 0.9649
Epoch [24/100], Train Loss: 0.0000, Val Loss: 0.0001 , train
acc :0.9679 , validation acc : 0.9659
Epoch [25/100], Train Loss: 0.0000, Val Loss: 0.0001 , train
acc :0.9689 , validation acc : 0.9670
Epoch [26/100], Train Loss: 0.0004, Val Loss: 0.0000 , train
acc :0.9649 , validation acc : 0.9630
Epoch [27/100], Train Loss: 0.0001, Val Loss: 0.0001 , train
acc :0.9695 , validation acc : 0.9673
Epoch [28/100], Train Loss: 0.0001, Val Loss: 0.0001 , train
acc :0.9701 , validation acc : 0.9684
Epoch [29/100], Train Loss: 0.0001, Val Loss: 0.0001 , train
acc :0.9735 , validation acc : 0.9714
Epoch [30/100], Train Loss: 0.0001, Val Loss: 0.0001 , train
acc :0.9745 , validation acc : 0.9723
Epoch [31/100], Train Loss: 0.0008, Val Loss: 0.0001 , train
acc :0.9695 , validation acc : 0.9674
Stopping early!
0.9576

model2 = FeedforwardNN(hidden_size1=265, hidden_size2=128)  # make a
new neural network
criterion = nn.CrossEntropyLoss()  # Cross-entropy loss
optimizer = optim.SGD(model2.parameters(), lr=0.01)
train_loss2,val_loss2,train_acc2,val_acc2=train_model(model2,
train_loader, val_loader, criterion, optimizer, epochs=100)
```

```
correct = 0
total = 0
model2.eval()  # Set the model to evaluation mode
with torch.no_grad():
    for images, labels in test_loader:
        outputs = model2(images)
        _, predicted = torch.max(outputs, 1)  # Get predicted class
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
test_accuracy=correct/total # till now this is the best test accuracy
print(test_accuracy)
if test_accuracy>besttest_accuracy:
  best_model=model2
  besttest_accuracy=test_accuracy
```

```
Epoch [1/100], Train Loss: 0.0007, Val Loss: 0.0016 , train
acc :0.8651 , validation acc : 0.8645
Epoch [2/100], Train Loss: 0.0008, Val Loss: 0.0009 , train
acc :0.8892 , validation acc : 0.8886
Epoch [3/100], Train Loss: 0.0005, Val Loss: 0.0006 , train
acc :0.9039 , validation acc : 0.9034
Epoch [4/100], Train Loss: 0.0001, Val Loss: 0.0003 , train
acc :0.9182 , validation acc : 0.9181
Epoch [5/100], Train Loss: 0.0002, Val Loss: 0.0003 , train
acc :0.9256 , validation acc : 0.9253
Epoch [6/100], Train Loss: 0.0001, Val Loss: 0.0003 , train
acc :0.9296 , validation acc : 0.9288
Epoch [7/100], Train Loss: 0.0000, Val Loss: 0.0003 , train
acc :0.9354 , validation acc : 0.9348
Epoch [8/100], Train Loss: 0.0001, Val Loss: 0.0002 , train
acc :0.9389 , validation acc : 0.9385
Epoch [9/100], Train Loss: 0.0003, Val Loss: 0.0001 , train
acc :0.9378 , validation acc : 0.9369
Epoch [10/100], Train Loss: 0.0004, Val Loss: 0.0002 , train
acc :0.9476 , validation acc : 0.9466
Epoch [11/100], Train Loss: 0.0002, Val Loss: 0.0002 , train
acc :0.9520 , validation acc : 0.9506
Epoch [12/100], Train Loss: 0.0003, Val Loss: 0.0001 , train
acc :0.9541 , validation acc : 0.9530
Epoch [13/100], Train Loss: 0.0001, Val Loss: 0.0001 , train
acc :0.9581 , validation acc : 0.9570
Epoch [14/100], Train Loss: 0.0003, Val Loss: 0.0003 , train
acc :0.9544 , validation acc : 0.9531
Stopping early!
0.9501
```

```
model3=FeedforwardNN2()
criterion = nn.CrossEntropyLoss()  # Cross-entropy loss
optimizer = optim.SGD(model3.parameters(), lr=0.01)
train_loss3,val_loss3,train_acc3,val_acc3=train_model(model3,
train_loader, val_loader, criterion, optimizer, epochs=100)
correct = 0
total = 0
model3.eval()  # Set the model to evaluation mode
with torch.no_grad():
    for images, labels in test_loader:
        outputs = model3(images)
        _, predicted = torch.max(outputs, 1)  # Get predicted class
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
test_accuracy=correct/total # till now this is the best test accuracy
print(test_accuracy)
if test_accuracy>besttest_accuracy:
  best_model=model3
  besttest_accuracy=test_accuracy
```

```
Epoch [1/100], Train Loss: 0.0009, Val Loss: 0.0040 , train
acc :0.7455 , validation acc : 0.7445
Epoch [2/100], Train Loss: 0.0007, Val Loss: 0.0017 , train
acc :0.8527 , validation acc : 0.8525
Epoch [3/100], Train Loss: 0.0007, Val Loss: 0.0011 , train
acc :0.8680 , validation acc : 0.8676
Epoch [4/100], Train Loss: 0.0002, Val Loss: 0.0004 , train
acc :0.9071 , validation acc : 0.9067
Epoch [5/100], Train Loss: 0.0001, Val Loss: 0.0004 , train
acc :0.9176 , validation acc : 0.9174
Epoch [6/100], Train Loss: 0.0001, Val Loss: 0.0005 , train
acc :0.9229 , validation acc : 0.9224
Epoch [7/100], Train Loss: 0.0003, Val Loss: 0.0004 , train
acc :0.9276 , validation acc : 0.9268
Epoch [8/100], Train Loss: 0.0001, Val Loss: 0.0004 , train
acc :0.9408 , validation acc : 0.9401
Epoch [9/100], Train Loss: 0.0001, Val Loss: 0.0003 , train
acc :0.9410 , validation acc : 0.9401
Epoch [10/100], Train Loss: 0.0001, Val Loss: 0.0003 , train
acc :0.9507 , validation acc : 0.9497
Epoch [11/100], Train Loss: 0.0000, Val Loss: 0.0003 , train
acc :0.9539 , validation acc : 0.9527
Epoch [12/100], Train Loss: 0.0002, Val Loss: 0.0001 , train
acc :0.9518 , validation acc : 0.9501
Epoch [13/100], Train Loss: 0.0001, Val Loss: 0.0014 , train
acc :0.9232 , validation acc : 0.9224
Epoch [14/100], Train Loss: 0.0001, Val Loss: 0.0001 , train
acc :0.9617 , validation acc : 0.9603
Epoch [15/100], Train Loss: 0.0000, Val Loss: 0.0002 , train
acc :0.9665 , validation acc : 0.9649
Epoch [16/100], Train Loss: 0.0000, Val Loss: 0.0002 , train
acc :0.9686 , validation acc : 0.9668
Epoch [17/100], Train Loss: 0.0002, Val Loss: 0.0002 , train
acc :0.9470 , validation acc : 0.9454
```

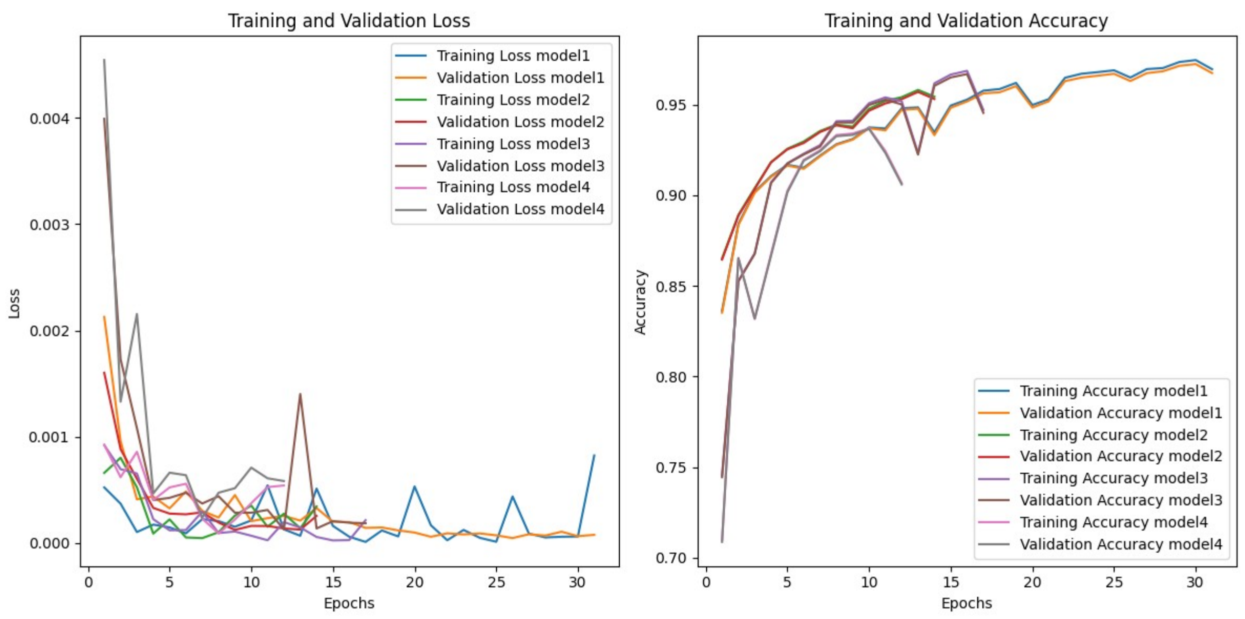
```
Stopping early!
0.941
```

```python
model4=FeedforwardNN2(hidden_size1=64,
hidden_size2=32,hidden_size3=16)
criterion = nn.CrossEntropyLoss()  # Cross-entropy loss
optimizer = optim.SGD(model4.parameters(), lr=0.01)
train_loss4,val_loss4,train_acc4,val_acc4=train_model(model4,
train_loader, val_loader, criterion, optimizer, epochs=100)
correct = 0
total = 0
model4.eval()  # Set the model to evaluation mode
with torch.no_grad():
    for images, labels in test_loader:
        outputs = model4(images)
        _, predicted = torch.max(outputs, 1)  # Get predicted class
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
test_accuracy=correct/total # till now this is the best test accuracy
print(test_accuracy)
if test_accuracy>besttest_accuracy:
  best_model=model4
  besttest_accuracy=test_accuracy
```

```
Epoch [1/100], Train Loss: 0.0009, Val Loss: 0.0045 , train
acc :0.7097 , validation acc : 0.7087
Epoch [2/100], Train Loss: 0.0006, Val Loss: 0.0013 , train
acc :0.8654 , validation acc : 0.8652
Epoch [3/100], Train Loss: 0.0009, Val Loss: 0.0022 , train
acc :0.8317 , validation acc : 0.8319
Epoch [4/100], Train Loss: 0.0004, Val Loss: 0.0005 , train
acc :0.8675 , validation acc : 0.8666
Epoch [5/100], Train Loss: 0.0005, Val Loss: 0.0007 , train
acc :0.9025 , validation acc : 0.9016
Epoch [6/100], Train Loss: 0.0006, Val Loss: 0.0006 , train
acc :0.9194 , validation acc : 0.9190
Epoch [7/100], Train Loss: 0.0002, Val Loss: 0.0002 , train
acc :0.9247 , validation acc : 0.9242
Epoch [8/100], Train Loss: 0.0001, Val Loss: 0.0005 , train
acc :0.9332 , validation acc : 0.9325
Epoch [9/100], Train Loss: 0.0002, Val Loss: 0.0005 , train
acc :0.9340 , validation acc : 0.9333
Epoch [10/100], Train Loss: 0.0004, Val Loss: 0.0007 , train
acc :0.9369 , validation acc : 0.9365
Epoch [11/100], Train Loss: 0.0005, Val Loss: 0.0006 , train
acc :0.9244 , validation acc : 0.9233
Epoch [12/100], Train Loss: 0.0005, Val Loss: 0.0006 , train
acc :0.9067 , validation acc : 0.9060
Stopping early!
0.9073
```

```python
print(besttest_accuracy)

# Dynamically set epochs based on the lengths of the loss/accuracy
lists
epochs1 = range(1, len(train_loss1) + 1)
epochs2 = range(1, len(train_loss2) + 1)
epochs3 = range(1, len(train_loss3) + 1)
epochs4 = range(1, len(train_loss4) + 1)

plt.figure(figsize=(12, 6))

# Loss plot
plt.subplot(1, 2, 1)
plt.plot(epochs1, train_loss1, label="Training Loss model1")
plt.plot(epochs1, val_loss1, label="Validation Loss model1")
plt.plot(epochs2, train_loss2, label="Training Loss model2")
plt.plot(epochs2, val_loss2, label="Validation Loss model2")
plt.plot(epochs3, train_loss3, label="Training Loss model3")
plt.plot(epochs3, val_loss3, label="Validation Loss model3")
plt.plot(epochs4, train_loss4, label="Training Loss model4")
plt.plot(epochs4, val_loss4, label="Validation Loss model4")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Training and Validation Loss")
plt.legend()

# Accuracy plot
plt.subplot(1, 2, 2)
plt.plot(epochs1, train_acc1, label="Training Accuracy model1")
plt.plot(epochs1, val_acc1, label="Validation Accuracy model1")
plt.plot(epochs2, train_acc2, label="Training Accuracy model2")
plt.plot(epochs2, val_acc2, label="Validation Accuracy model2")
plt.plot(epochs3, train_acc3, label="Training Accuracy model3")
plt.plot(epochs3, val_acc3, label="Validation Accuracy model3")
plt.plot(epochs4, train_acc4, label="Training Accuracy model4")
plt.plot(epochs4, val_acc4, label="Validation Accuracy model4")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Training and Validation Accuracy")
plt.legend()

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

```
0.9785
```

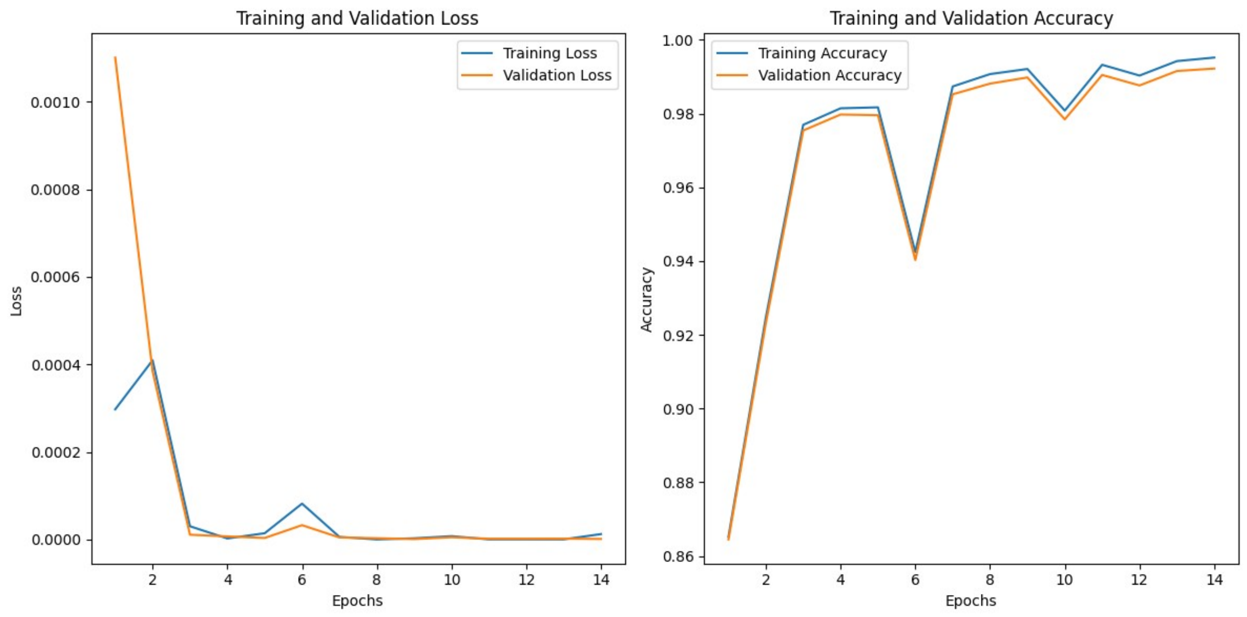


```
#making a model with the best of best values of each
def train_model(model, train_loader, val_loader, criterion, optimizer,
epochs=100):
    best_val_loss = float('inf')
    patience = 5
    epochs_without_improvement = 0
    train_losses,val_losses,train_acc,val_acc = [], [],[],[]
    for epoch in range(epochs):  # loop for maximum 100 iterations
        model.train() # training the model
        train_loss = 0.0 #initialize loss for each epoch
        for images, labels in train_loader:  # loop on training data
            # Zero the gradients
            optimizer.zero_grad()  # start with zero gradient
            # Forward pass
            outputs = model(images)  # get the expected output of the
images depending on model
            loss = criterion(outputs, labels)  # calculate the loss
for the iteration
            # Backward pass
            loss.backward()  # calculate the gradient of the loss
function
            optimizer.step() #make a stochastic gradient descent step
            train_loss += loss.item()
            train_loss /= len(train_loader) # Average training loss
        train_losses.append(train_loss)
        correct = 0
        total = 0
        model.eval()  # Set the model to evaluation mode
        with torch.no_grad():
            for images, labels in train_loader:
                outputs = model(images)
                _, predicted = torch.max(outputs, 1)  # Get predicted
class
                #predicted: stores the indices of the maximum values,
corresponding to the predicted class labels.
                total += labels.size(0) #A counter that accumulates
the number of samples processed so far during the loop.
                correct += (predicted == labels).sum().item()
        trainacc=correct/total
        train_acc.append(trainacc)
        # Validation phase
        model.eval()
        val_loss = 0.0
        with torch.no_grad(): #statement in PyTorch is used to
temporarily disable gradient computation
            for images, labels in val_loader:
                outputs = model(images)
                _, predicted = torch.max(outputs, 1)  # Get predicted
class
                total += labels.size(0)
                correct += (predicted == labels).sum().item()
                loss = criterion(outputs, labels)
                val_loss += loss.item()
                val_loss /= len(val_loader)
            # Average validation loss
        val_losses.append(val_loss)
        validacc=correct/total
        val_acc.append(validacc)
        print(f"Epoch [{epoch+1}/{epochs}], Train Loss:
{train_loss:.4f}, Val Loss: {val_loss:.4f} , train acc :{trainacc:.4f}
, validation acc : {validacc:.4f}")
        #early exit process happen
        if val_loss < best_val_loss:
            best_val_loss = val_loss
            epochs_without_improvement = 0  # Reset patience counter
        else:
            epochs_without_improvement += 1
        if epochs_without_improvement >= patience:
            print("Stopping early!")
            return train_losses, val_losses ,train_acc,val_acc
    return train_losses, val_losses ,train_acc,val_acc
BATCH_SIZE=32
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE,
shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=BATCH_SIZE,
shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE,
shuffle=False)
model2 = FeedforwardNN(hidden_size1=265, hidden_size2=128)  # make a
```

```
new neural network
criterion = nn.CrossEntropyLoss()  # Cross-entropy loss
optimizer = optim.SGD(model2.parameters(), lr=0.1)
train_losses,val_losses,train_accuracy,val_accuracy=train_model(model2
, train_loader, val_loader, criterion, optimizer, epochs=100)
model2.eval()
test_loss = 0
correct = 0
total = 0
all_labels = []
all_preds = []
with torch.no_grad():
    for images, labels in test_loader:
        outputs = model2(images)
        loss = criterion(outputs, labels)
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
        all_labels.extend(labels.numpy())
        all_preds.extend(predicted.numpy())
test_loss /= len(test_loader)
test_accuracy = 100 * (correct / total)
print(f"Test Loss: {test_loss:.4f}, Test Accuracy: {test_accuracy:.2f}
%")
conf_matrix = confusion_matrix(all_labels, all_preds)
class_report = classification_report(all_labels, all_preds)
print("Confusion Matrix:\n", conf_matrix)
print("\nClassification Report:\n", class_report)
epochs = range(1, len(train_losses) + 1)
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(epochs, train_losses, label="Training Loss")
plt.plot(epochs, val_losses, label="Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Training and Validation Loss")
plt.legend()
plt.subplot(1, 2, 2)
plt.plot(epochs, train_accuracy, label="Training Accuracy")
plt.plot(epochs, val_accuracy, label="Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Training and Validation Accuracy")
plt.legend()
plt.tight_layout()
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/torchvision/transforms/
functional.py:154: UserWarning: The given NumPy array is not writable,
and PyTorch does not support non-writable tensors. This means writing
to this tensor will result in undefined behavior. You may want to copy
the array to protect its data or make it writable before converting it
to a tensor. This type of warning will be suppressed for the rest of
this program. (Triggered internally at
../torch/csrc/utils/tensor_numpy.cpp:206.)
  img = torch.from_numpy(pic.transpose((2, 0, 1))).contiguous()

Epoch [1/100], Train Loss: 0.0003, Val Loss: 0.0011 , train
acc :0.8652 , validation acc : 0.8645
Epoch [2/100], Train Loss: 0.0004, Val Loss: 0.0004 , train
acc :0.9247 , validation acc : 0.9232
Epoch [3/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9770 , validation acc : 0.9754
Epoch [4/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9814 , validation acc : 0.9798
Epoch [5/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9817 , validation acc : 0.9796
Epoch [6/100], Train Loss: 0.0001, Val Loss: 0.0000 , train
acc :0.9424 , validation acc : 0.9403
Epoch [7/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9874 , validation acc : 0.9852
Epoch [8/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9907 , validation acc : 0.9881
Epoch [9/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9921 , validation acc : 0.9898
Epoch [10/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9808 , validation acc : 0.9784
Epoch [11/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9932 , validation acc : 0.9905
Epoch [12/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9903 , validation acc : 0.9876
Epoch [13/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9942 , validation acc : 0.9916
Epoch [14/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9952 , validation acc : 0.9922
Stopping early!
Test Loss: 0.0000, Test Accuracy: 97.81%
Confusion Matrix:
 [[ 964    0    4    1    1    3    3    1    3    0]
 [   0 1126    1    2    0    0    1    0    5    0]
 [   2    0 1017    4    1    0    2    5    1    0]
 [   0    0    2  997    0    2    0    4    3    2]
 [   0    1    3    0  963    2    5    4    0    4]
 [   2    0    0   20    1  861    3    2    1    2]
 [   2    3    1    0    2   11  937    0    2    0]
 [   1    8    5    5    1    0    0 1001    2    5]
 [   3    0    3    7    1    6    0    3  950    1]
 [   0    3    1    9   13   10    1    5    2  965]]
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.98      0.99       980
           1       0.99      0.99      0.99      1135
           2       0.98      0.99      0.98      1032
           3       0.95      0.99      0.97      1010
           4       0.98      0.98      0.98       982
           5       0.96      0.97      0.96       892
           6       0.98      0.98      0.98       958
           7       0.98      0.97      0.98      1028
           8       0.98      0.98      0.98       974
           9       0.99      0.96      0.97      1009

    accuracy                           0.98     10000
   macro avg       0.98      0.98      0.98     10000
weighted avg       0.98      0.98      0.98     10000
```

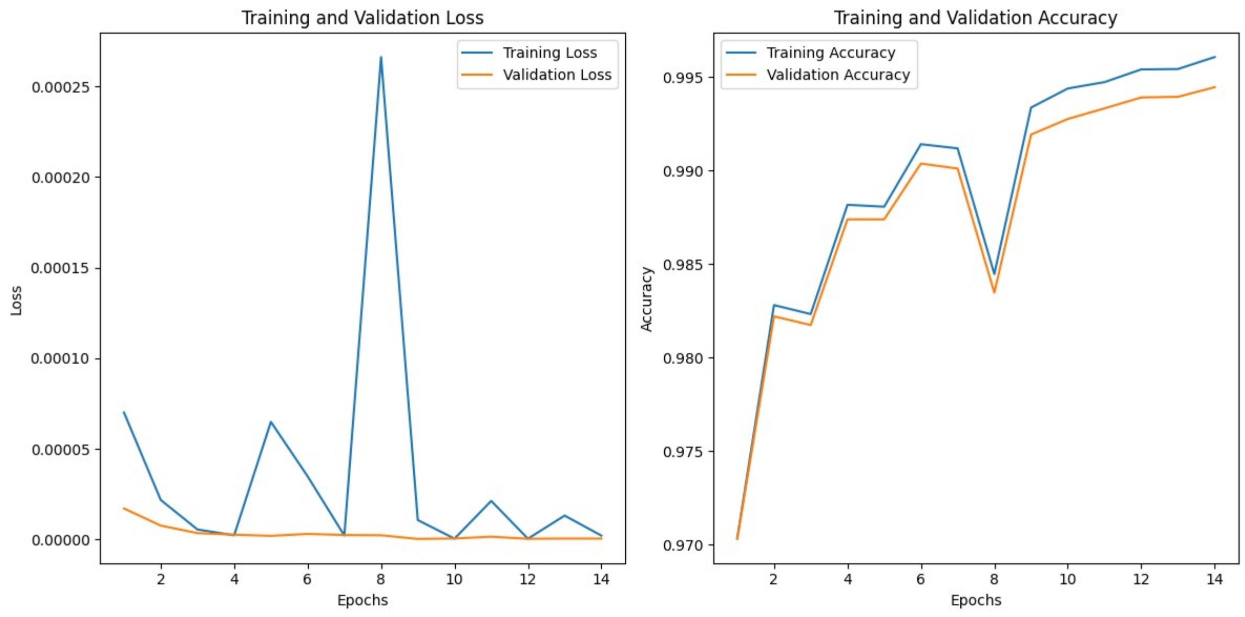


```
class CNNModel(nn.Module):
    def __init__(self, output_size=10):
        super(CNNModel, self).__init__()
        # Convolutional Layer 1
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=32,
kernel_size=3, padding=1)
        self.norm1 = nn.LayerNorm([32, 28, 28])  # Layer Normalization
for Conv1 output
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)  # Max
Pooling
        # Convolutional Layer 2
        self.conv2 = nn.Conv2d(in_channels=32, out_channels=64,
kernel_size=3, padding=1)
        self.norm2 = nn.LayerNorm([64, 14, 14])  # Layer Normalization
for Conv2 output
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)  # Max
Pooling
        # Fully connected layers
        self.fc1 = nn.Linear(64 * 7 * 7, 128)
        self.fc2 = nn.Linear(128, output_size)
        # Dropout layer
        self.dropout = nn.Dropout(p=0.5)  # Dropout with 50%
probability
    def forward(self, x):
        x = self.pool1(self.norm1(self.conv1(x)))  # Apply Conv1,
Norm1, Pool1
        x = self.pool2(self.norm2(self.conv2(x)))  # Apply Conv2,
Norm2, Pool2
        x = x.view(x.size(0), -1)  # Flatten the output for the fully
connected layers
        x = self.fc1(x)  # First fully connected layer
        x = self.dropout(x)  # Apply Dropout during training
        x = self.fc2(x)  # Output layer
        return x
def train_model_conv(model, train_loader, val_loader, criterion,
optimizer, epochs=100):
    train_losses, val_losses, train_acc, val_acc = [], [], [], []
    patience=5
    for epoch in range(epochs):
        model.train()  # Set model to training mode
        train_loss = 0.0
        correct = 0
        total = 0
        for images, labels in train_loader:
            optimizer.zero_grad()
            outputs = model(images)  # Forward pass
            loss = criterion(outputs, labels)
            loss.backward()  # Backward pass
            optimizer.step()  # Optimize parameters
            train_loss += loss.item()
            _, predicted = torch.max(outputs, 1)  # Get predicted
class
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
        train_accuracy = 100 * correct / total
        train_losses.append(train_loss / len(train_loader))
        train_acc.append(train_accuracy)
        # Validation phase (without Dropout)
        model.eval()  # Set model to evaluation mode (turn off
dropout)
```

```
        val_loss = 0.0
        correct = 0
        total = 0
        with torch.no_grad():  # No gradient calculation for
validation
            for images, labels in val_loader:
                outputs = model(images)
                loss = criterion(outputs, labels)
                val_loss += loss.item()
                _, predicted = torch.max(outputs, 1)  # Get predicted
class
                total += labels.size(0)
                correct += (predicted == labels).sum().item()
        val_accuracy = 100 * correct / total
        val_losses.append(val_loss / len(val_loader))
        val_acc.append(val_accuracy)
        print(f"Epoch {epoch+1}/{epochs} | Train Loss:
{train_loss/len(train_loader):.4f} | "
              f"Train Accuracy: {train_accuracy:.2f}% | Val Loss:
{val_loss/len(val_loader):.4f} | "
              f"Val Accuracy: {val_accuracy:.2f}%")
        if val_loss < best_val_loss:
            best_val_loss = val_loss
            epochs_without_improvement = 0  # Reset patience counter
        else:
            epochs_without_improvement += 1
        if epochs_without_improvement > patience:
            print("Stopping early!")
            return train_losses, val_losses ,train_acc,val_acc
model=CNNModel()
criterion = nn.CrossEntropyLoss()  # Cross-entropy loss
optimizer = optim.SGD(model.parameters(), lr=0.01)
train_losses, val_losses,train_accuracy,val_accuracy =
train_model(model, train_loader, val_loader, criterion, optimizer,
epochs=100)
plot_fn()
```

```
Epoch [1/100], Train Loss: 0.0001, Val Loss: 0.0000 , train
acc :0.9703 , validation acc : 0.9703
Epoch [2/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9828 , validation acc : 0.9822
Epoch [3/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9823 , validation acc : 0.9817
Epoch [4/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9882 , validation acc : 0.9874
Epoch [5/100], Train Loss: 0.0001, Val Loss: 0.0000 , train
acc :0.9881 , validation acc : 0.9874
Epoch [6/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9914 , validation acc : 0.9904
```

```
Epoch [7/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9912 , validation acc : 0.9901
Epoch [8/100], Train Loss: 0.0003, Val Loss: 0.0000 , train
acc :0.9845 , validation acc : 0.9835
Epoch [9/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9934 , validation acc : 0.9919
Epoch [10/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9944 , validation acc : 0.9928
Epoch [11/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9947 , validation acc : 0.9933
Epoch [12/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9954 , validation acc : 0.9939
Epoch [13/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9954 , validation acc : 0.9939
Epoch [14/100], Train Loss: 0.0000, Val Loss: 0.0000 , train
acc :0.9961 , validation acc : 0.9944
Stopping early!
0.9881
```



1.Convolutional Layers: Use nn.Conv2d for convolutional layers, which are better suited for image data like MNIST. 2.Max Pooling Layers: Use nn.MaxPool2d to reduce the spatial dimensions of feature maps. 3.Dropout Layers: Use nn.Dropout to randomly set some of the activations to zero during training, helping with regularization and preventing overfitting. 4.Layer Normalization: Apply nn.LayerNorm to normalize the output of a layer, which can help with convergence during training.

1.Convolutional Layers (nn.Conv2d):

conv1: Takes 1 input channel (grayscale image), outputs 32 channels, and uses a 3x3 kernel with padding to preserve the spatial dimensions. conv2: Takes 32 input channels (from the previous convolutional layer) and outputs 64 channels.

2.Max Pooling (nn.MaxPool2d):

Pooling is used to reduce the spatial dimensions after each convolutional block, which also helps reduce computational complexity and prevents overfitting.

3.Layer Normalization (nn.LayerNorm):

norm1 and norm2: Normalize the output of the convolutional layers to stabilize training and improve convergence. 4.Dropout Layer (nn.Dropout):

After the first fully connected layer (fc1), dropout is applied with a probability of 0.5 to randomly set half of the neurons to zero during training. This helps prevent overfitting by making the model more robust.

Training Performance:

With Dropout: The model should be less prone to overfitting, meaning it will generalize better to unseen data. However, dropout may slightly slow down training as some neurons are randomly dropped during each update. With Layer Normalization: The model will likely converge faster due to stabilized training. Layer normalization helps with issues like vanishing/exploding gradients by ensuring that activations are normalized, thus improving the flow of gradients.
Final Performance:

Without Dropout: The model might achieve slightly higher accuracy on the training set, but it could overfit and perform poorly on the validation or test set. Without Layer Normalization: The model might converge more slowly or get stuck in poor local minima. Training might also be more sensitive to the learning rate.