To prepare our CIFAR10 dataset for training, we apply data augmentation techniques and transforms. Specifically, we use the following techniques:

Random Crop: This technique creates a random subset of an original image. We use this to randomly crop 32x32-pixel patches from the original images, which helps our model learn to recognize objects that might appear anywhere in the image.

Random Horizontal Flip: This technique flips an image horizontally with a probability p. We use this to further augment our dataset and make our model more robust to variations in the orientation of objects.

To Tensor: We use the To Tensor transform to convert our images to PyTorch tensors and scale the pixel values by 255.

Normalize: We apply the Normalize transform to the tensor images so that their values have a mean of 0.0 and a standard deviation of 1.0. This helps our model learn from the input data more effectively.

We download the CIFAR10 dataset to the root directory ./data. To ensure that we get the same validation set each time we run the code, we set PyTorch's random number generator to a seed value of 17. We also import the datasets and convert the images into PyTorch tensors.

```
import multiprocessing
import torchvision
import torchvision.transforms as transforms
import numpy as np
from torch.utils.data import DataLoader
import torch
torch.manual_seed(17)

from torchsummary import summary
from tqdm import tqdm
import matplotlib.pyplot as plt

class FetchDataset:

    def __init__(self, dataset="CIFAR10", batch_size=64):
        print("Initializing fetching %s dataset using torchvision"%
(dataset))
        # check if the dataset exists in torchvision
        self.datasetObject =
torchvision.datasets.__dict__.get(dataset, None)
        if self.datasetObject == None:
            raise Exception("Dataset %s might not be in torchvision."%
(dataset))
        self.batch_size = batch_size
        self.transformers_training = []
        self.transformers_testing = []
```

```
        # set number of workers available for multiprocessing
        self.workersAvailable = min(multiprocessing.cpu_count(), 14)

    def dataAugmentation(self, size=32, padding=3):
        # add data augmentation transforms to the training set

self.transformers_training.append(transforms.RandomHorizontalFlip())

self.transformers_training.append(transforms.RandomCrop(size=size,
padding=padding))

self.transformers_training.append(transforms.functional.equalize)

self.transformers_testing.append(transforms.functional.equalize)

    def __addToTensor(self):
        # add ToTensor transform to the training and testing sets
        self.transformers_training.append(transforms.ToTensor())
        self.transformers_testing.append(transforms.ToTensor())

    def addNormalizer(self):
        self.__addToTensor()
        # load training set to compute mean and standard deviation
        dataset_training = self.datasetObject(root="./data",
train=True, download=True)
        data_train = dataset_training.data/255.0
        mean = data_train.mean(axis=(0, 1, 2))
        std = data_train.std(axis=(0, 1, 2))
        # add Normalize transform to the training and testing sets

self.transformers_training.append(transforms.Normalize(mean=mean,
std=std))

self.transformers_testing.append(transforms.Normalize(mean=mean,
std=std))


    def getLoaders(self):
        if len(self.transformers_training) == 0:
            self.__addToTensor()
        # create data loaders with the defined batch size,
transformers and number of workers
        dataset_training = self.datasetObject(root="./data",
train=True, download=True,
transform=transforms.Compose(self.transformers_training))
        dataset_testing = self.datasetObject(root="./data",
train=False, download=True,
transform=transforms.Compose(self.transformers_testing))
        load_train = DataLoader(dataset_training,
batch_size=self.batch_size, shuffle=True,
```

```
num_workers=self.workersAvailable)
        load_test = DataLoader(dataset_testing,
batch_size=self.batch_size, shuffle=False,
num_workers=self.workersAvailable)
        # return the training and testing data loaders
        return load_train, load_test

# create a new instance of FetchDataset for the CIFAR10 dataset with
batch size of 128
df = FetchDataset(dataset="CIFAR10", batch_size=128)
# add data augmentation transforms to the training set with size 32
and padding 4
df.dataAugmentation(size=32, padding=4)
# add normalizing transforms to the training and testing sets
df.addNormalizer()
# get the training and testing data loaders
trainLoader, testLoader = df.getLoaders()
```

```
Initializing fetching CIFAR10 dataset using torchvision
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to
./data/cifar-10-python.tar.gz

100%|██████████| 170498071/170498071 [00:04<00:00, 39962218.40it/s]

Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
Files already downloaded and verified
```

Modified the basic resnet model from
https://github.com/kuangliu/pytorch-cifar/blob/master/models/resnet.py

• Ci, the number of channels in the ith layer. • Fi, the filter size in the ith layer • Ki, the kernel size in the ith skip connection • P, the pool size in the average pool laye

Using the following values for the Hyperparameters (design variables) in our architectures :

```
C=[64,128,128,256]
F=[3,3,3,3]
K=[1,1,1,1]
P=4
```

```
# Import necessary PyTorch modules
import torch.nn as nn
import torch.nn.functional as F

# Define the BasicBlock class, which is used to construct the layers
in the ResNet architecture
class BasicBlock(nn.Module):

    # Initialize the BasicBlock class
```

```
    def __init__(self, in_planes, planes, kernel_size, skip_kernel,
stride=1):
        super(BasicBlock, self).__init__()

        # Define the first convolutional layer
        self.conv1 = nn.Conv2d(in_planes, planes,
kernel_size=kernel_size, stride=stride, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(planes)

        # Define the second convolutional layer
        self.conv2 = nn.Conv2d(planes, planes,
kernel_size=kernel_size, stride=1, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(planes)

        # Define the shortcut connection, which is used to add the
output of the convolutional layers to the input
        self.shortcut = nn.Sequential()

        # If the stride is not 1 or the number of input planes is not
equal to the number of output planes,
        # define a convolutional layer and a batch normalization layer
for the shortcut connection
        if stride != 1 or in_planes != planes:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_planes, planes, kernel_size=skip_kernel,
stride=stride, bias=False),
                nn.BatchNorm2d(planes)
            )

    # Define the forward pass for the BasicBlock class
    def forward(self, x):
        # Apply the first convolutional layer, batch normalization,
and ReLU activation
        out = F.relu(self.bn1(self.conv1(x)))

        # Apply the second convolutional layer and batch normalization
        out = self.bn2(self.conv2(out))

        # Add the shortcut connection to the output of the
convolutional layers
        out += self.shortcut(x)

        # Apply the ReLU activation
        out = F.relu(out)

        return out

# Define the ResNet class, which is used to construct the ResNet
architecture
```

```python
class ResNet(nn.Module):

      # Initialize the ResNet class
      def __init__(self,N:int, B:list, C:list, F:list, K:list, P:int,
num_classes=10):
          super(ResNet, self).__init__()

          # Initialize the number of input planes
          self.in_planes = C[0]

          # Set the block to the BasicBlock class
          self.block = BasicBlock

          # Store the values of N, B, C, F, K, and P
          self.N, self.B, self.C, self.F, self.K, self.P= N, B, C, F,
K, P

          # Initialize a container for the layers
          self.layers = []

          # Set the stride for each layer
          self.S = [2] * N
          self.S[0] = 1

          # Calculate the input dimension for the output linear layer
          self.outLayerInSize = C[N-1]*(32//(P*2**(N-
1)))*(32//(P*2**(N-1)))


          # Print Model Config
          print("\n\nModel Config: "
              "\n-------------------------------------"
              "\nN (# Layers)\t:",self.N,
              "\nB (# Blocks)\t:",self.B,
              "\nC (# Channels)\t:",C,
              "\nF (Conv Kernel)\t:",F,
              "\nK (Skip Kernel)\t:",K,
              "\nP (Pool Kernel)\t:",P,)

          # Define the first convolution layer with 3 input channels,
C[0] output channels, F[0] kernel size,
          # stride of 1, padding of 1, and no bias

          self.conv1 = nn.Conv2d(3, C[0], kernel_size=F[0], stride=1,
padding=1, bias=False)
          # Define a batch normalization layer with C[0] channels

          self.bn1 = nn.BatchNorm2d(C[0])
                  # Define N residual blocks
```

```
        for i in range(N):
            # Dynamically create variable names for each residual
block using the exec() function

            exec("self.layer{} = self._make_layer(self.block,
self.C[{}], self.B[{}], self.F[{}], self.K[{}], self.S[{}])"\
                .format(i+1,i,i,i,i,i))
            # Append the residual block to the layers ModuleList
            exec("self.layers.append(self.layer{})".format(i+1))
                  # Define the final linear layer with input size of
outLayerInSize and output size of num_classes

        self.linear = nn.Linear(self.outLayerInSize, num_classes)


    def _make_layer(self, block, planes, num_blocks, kernel_size,
skip_kernel, stride):
        # Set stride for each block in the layer
        strides = [stride] + [1]*(num_blocks-1)
        layers = []
        for stride in strides:
            # Append each block to the layer with given arguments
            layers.append(block(self.in_planes, planes, kernel_size,
skip_kernel, stride))
            # Update the number of input planes for the next block
            self.in_planes = planes
        # Return a sequential module containing all the blocks in
the layer
        return nn.Sequential(*layers)


    def forward(self, x):
        # Apply the first convolutional layer followed by batch
normalization and ReLU activation
        out = F.relu(self.bn1(self.conv1(x)))
        # Apply all the blocks in the layer
        for layer in self.layers:
            out = layer(out)
        # Apply average pooling with kernel size self.P
        out = F.avg_pool2d(out, self.P)
        # Flatten the output tensor
        out = out.view(out.size(0), -1)
        # Apply the fully connected linear layer
        out = self.linear(out)
        # Return the final output
        return out

def resnet_model():
      # Define the parameters for the ResNet architecture
```

```
        B=[3,3,2,3]  # number of blocks in each layer
        C=[64,128,128,256]  # number of output channels in each layer
        F=[3,3,3,3]  # kernel size for each layer
        K=[1,1,1,1]  # skip kernel size for each layer
        P=4  # average pooling kernel size
        N=len(B)  # number of layers in the network
        # Return a new ResNet model with the defined parameters
        return ResNet(N, B, C, F, K, P)
```

Checking Device - If GPU is available, GPU will be used.

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

```
cuda
```

```
model = resnet_model()
model = model.to(device)
```

```
Model Config:
-------------------------------------
N (# Layers)     : 4
B (# Blocks)     : [3, 3, 2, 3]
C (# Channels)   : [64, 128, 128, 256]
F (Conv Kernel)  : [3, 3, 3, 3]
K (Skip Kernel)  : [1, 1, 1, 1]
P (Pool Kernel)  : 4
```

We run our model for 300 epochs, to find out the best possible accuracy. The accuracy becomes near about constant after it. We define our:

learning rate, weightDecay, type of optimizer to be used (we tried with Adam, Adagrad, AdaDelta), with Adadelta giving out the best accuracy.

The scheduler set the learning rate of each parameter group using a cosine annealing schedule

```
EPOCHS=300
globalBestAccuracy = 0.0  # initialize the global best accuracy to 0.0
train_loss = []  # list to store train loss
test_loss = []  # list to store test loss
train_accuracy = []  # list to store train accuracy
test_accuracy = []  # list to store test accuracy

# define the loss function as Cross Entropy Loss with sum reduction
loss_function = torch.nn.CrossEntropyLoss(reduction='sum')

learningRate = 0.01  # set the learning rate to 0.01
weightDecay = 0.0001  # set the weight decay to 0.0001
```

```
# define the optimizer as Adadelta with the above defined learning
rate and weight decay
optimizer = torch.optim.Adadelta(model.parameters(), lr=learningRate,
weight_decay=weightDecay)

# define the learning rate scheduler as Cosine Annealing LR with the
above defined optimizer, number of epochs, and minimum learning rate
scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer,
EPOCHS, eta_min=learningRate/10.0)

# print the model's evaluation mode
print(model.eval())

# calculate and print the total trainable parameters of the model
trainable_parameters = sum(p.numel() for p in model.parameters() if
p.requires_grad)
print("Total Trainable Parameters : %s"%(trainable_parameters))

# if the total number of trainable parameters exceeds 5 million, raise
an exception
if trainable_parameters > 5*(10**6):
    raise Exception("The total number of parameters exceeds 5
million")
```

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (shortcut): Sequential()
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
```

```
track_running_stats=True)
      (shortcut): Sequential()
    )
    (2): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (shortcut): Sequential()
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (shortcut): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2),
bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (shortcut): Sequential()
    )
    (2): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
```

```
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (shortcut): Sequential()
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (shortcut): Sequential(
        (0): Conv2d(128, 128, kernel_size=(1, 1), stride=(2, 2),
bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (shortcut): Sequential()
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (shortcut): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2),
bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
```

```
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (shortcut): Sequential()
    )
    (2): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (shortcut): Sequential()
    )
  )
  (linear): Linear(in_features=256, out_features=10, bias=True)
)
Total Trainable Parameters : 4935242
```

Model Training and Testing. Here we are training our model over 300 epochs and trying to find out the best accuracy our model provides for the CIFAR dataset.

```
# Define the function to train the model and return the updated model
and optimizer
def train(model, loader, optimizer):
    # Set the model to train mode
    model.train()
    # Clear the gradients of the optimizer
    optimizer.zero_grad()
    # Return the updated model and optimizer
    return model, optimizer

# Define the function to evaluate the model on the test set
def test(model, loader):
    # Set the model to evaluation mode
    return model.eval()

# Define the function to calculate the loss and accuracy for a given
loader and model
def getLoss(loader, model, optimizer, phase):
    # Initialize the running loss and correct count
```

```
    running_loss = 0.0
    running_correct = 0
    # Iterate through the loader
    for images, labels in loader:
        # Move the images and labels to the device
        images = images.to(device)
        labels = labels.to(device)
        # Forward pass
        output = model(images)
        # Calculate the loss
        loss = loss_function(output, labels)
        # Calculate the predicted labels
        predicted_labels = torch.argmax(output, dim=1)
        # Update the running loss and correct count
        running_loss += loss.item()
        running_correct += torch.sum(predicted_labels ==
labels).float().item()
        # If the phase is "train", backpropagate the loss and update
the optimizer
        if phase == "train":
            loss.backward()
            optimizer.step()
            optimizer.zero_grad()
    # Calculate the epoch loss and accuracy
    epoch_loss = running_loss / len(loader.dataset)
    epoch_acc = running_correct / len(loader.dataset)
    # Return the epoch loss and accuracy
    return epoch_loss, epoch_acc

# Iterate through the epochs
for i in tqdm(range(EPOCHS)):
    # Iterate through the phases (train and test)
    for phase in ['train', 'test']:
        # If the phase is "train", set the loader and call the train
function
        if phase == "train":
            loader = trainLoader
            model, optimizer = train(model, loader, optimizer)
        # If the phase is "test", set the loader and call the test
function
        else:
            loader = testLoader
            model = test(model, loader)
        # Calculate the loss and accuracy for the current phase
        epoch_loss, epoch_acc = getLoss(loader, model, optimizer,
phase)
        # If the phase is "train", update the scheduler and append the
results to the train lists
        if phase == "train":
            scheduler.step()
```

```
            train_loss.append(epoch_loss)
            train_accuracy.append(epoch_acc)
        # If the phase is "test", append the results to the test lists
and update the global best accuracy
        else:
            test_loss.append(epoch_loss)
            test_accuracy.append(epoch_acc)
            globalBestAccuracy = max(globalBestAccuracy, epoch_acc)
    # Print the results for the current epoch
    print("Training Loss : %s, Testing Loss : %s, Training Accuracy :
%s, Testing Accuracy : %s" \
          % (train_loss[-1], test_loss[-1], train_accuracy[-1],
test_accuracy[-1]))
```

```
  0%|          | 1/300 [00:58<4:51:43, 58.54s/it]

Training Loss : 1.750513836517334, Testing Loss : 1.6276763652801514,
Training Accuracy : 0.36502, Testing Accuracy : 0.4239

  1%|          | 2/300 [01:48<4:26:46, 53.71s/it]

Training Loss : 1.3847110874938964, Testing Loss : 1.3207223026275634,
Training Accuracy : 0.50854, Testing Accuracy : 0.5412

  1%|          | 3/300 [02:39<4:18:14, 52.17s/it]

Training Loss : 1.2345377491760254, Testing Loss : 1.224829564857483,
Training Accuracy : 0.5668, Testing Accuracy : 0.57

  1%|▏         | 4/300 [03:29<4:13:25, 51.37s/it]

Training Loss : 1.113235101928711, Testing Loss : 1.2847113328933715,
Training Accuracy : 0.60824, Testing Accuracy : 0.5776

  2%|▏         | 5/300 [04:19<4:10:50, 51.02s/it]

Training Loss : 1.0122601741027832, Testing Loss : 1.0716696825027465,
Training Accuracy : 0.64378, Testing Accuracy : 0.6294

  2%|▏         | 6/300 [05:10<4:09:33, 50.93s/it]

Training Loss : 0.9275491195678711, Testing Loss : 0.9625132943153382,
Training Accuracy : 0.6759, Testing Accuracy : 0.6653

  2%|▏         | 7/300 [06:01<4:08:54, 50.97s/it]

Training Loss : 0.8532541835784913, Testing Loss : 0.9616029819488525,
Training Accuracy : 0.70206, Testing Accuracy : 0.6814

  3%|▎         | 8/300 [06:52<4:08:33, 51.07s/it]

Training Loss : 0.7926426062011719, Testing Loss : 0.9885334470748901,
Training Accuracy : 0.72542, Testing Accuracy : 0.6716
```

```
  3%|▎         | 9/300 [07:43<4:07:10, 50.96s/it]

Training Loss : 0.7574244557189942, Testing Loss : 0.9268176073074341,
Training Accuracy : 0.73672, Testing Accuracy : 0.698

  3%|▎         | 10/300 [08:34<4:05:59, 50.90s/it]

Training Loss : 0.7071595803070069, Testing Loss : 0.8291011363983154,
Training Accuracy : 0.75556, Testing Accuracy : 0.7223

  4%|▎         | 11/300 [09:25<4:05:24, 50.95s/it]

Training Loss : 0.6629062732696533, Testing Loss : 0.7362963767528534,
Training Accuracy : 0.77008, Testing Accuracy : 0.7473

  4%|▍         | 12/300 [10:16<4:04:40, 50.97s/it]

Training Loss : 0.6248297144317627, Testing Loss : 0.7610584497451782,
Training Accuracy : 0.7836, Testing Accuracy : 0.751

  4%|▍         | 13/300 [11:07<4:03:51, 50.98s/it]

Training Loss : 0.6092548852539063, Testing Loss : 0.6856428800582886,
Training Accuracy : 0.78924, Testing Accuracy : 0.7674

  5%|▍         | 14/300 [11:58<4:02:59, 50.98s/it]

Training Loss : 0.5650048509216309, Testing Loss : 0.6995527620315551,
Training Accuracy : 0.80508, Testing Accuracy : 0.7668

  5%|▌         | 15/300 [12:48<4:01:01, 50.74s/it]

Training Loss : 0.5498452284240722, Testing Loss : 0.6795084643363952,
Training Accuracy : 0.81002, Testing Accuracy : 0.7694

  5%|▌         | 16/300 [13:38<3:58:56, 50.48s/it]

Training Loss : 0.526840435409546, Testing Loss : 0.6165720776557923,
Training Accuracy : 0.81878, Testing Accuracy : 0.7894

  6%|▌         | 17/300 [14:28<3:57:44, 50.40s/it]

Training Loss : 0.4960941240692139, Testing Loss : 0.6122942449092865,
Training Accuracy : 0.82982, Testing Accuracy : 0.7907

  6%|▌         | 18/300 [15:19<3:56:52, 50.40s/it]

Training Loss : 0.4880392674255371, Testing Loss : 0.6232776842594147,
Training Accuracy : 0.83204, Testing Accuracy : 0.7882

  6%|▋         | 19/300 [16:09<3:55:45, 50.34s/it]

Training Loss : 0.46933479026794434, Testing Loss : 0.589550674533844,
Training Accuracy : 0.83782, Testing Accuracy : 0.8038
```

```
  7%|▋         | 20/300 [17:00<3:56:34, 50.70s/it]
Training Loss : 0.4388230741882324, Testing Loss : 0.5901583492279052,
Training Accuracy : 0.84762, Testing Accuracy : 0.7997
  7%|▋         | 21/300 [17:53<3:58:35, 51.31s/it]
Training Loss : 0.4277646355819702, Testing Loss : 0.6005998805999756,
Training Accuracy : 0.85, Testing Accuracy : 0.7997
  7%|▋         | 22/300 [18:45<3:58:33, 51.49s/it]
Training Loss : 0.40895013946533204, Testing Loss :
0.5682986152172088, Training Accuracy : 0.85876, Testing Accuracy :
0.8125
  8%|▊         | 23/300 [19:37<3:58:45, 51.72s/it]
Training Loss : 0.3925004592895508, Testing Loss : 0.5572887885093689,
Training Accuracy : 0.86442, Testing Accuracy : 0.815
  8%|▊         | 24/300 [20:29<3:57:47, 51.69s/it]
Training Loss : 0.3897362612533569, Testing Loss : 0.550638566160202,
Training Accuracy : 0.86504, Testing Accuracy : 0.819
  8%|▊         | 25/300 [21:20<3:56:29, 51.60s/it]
Training Loss : 0.3708708860015869, Testing Loss : 0.5533961500644684,
Training Accuracy : 0.87206, Testing Accuracy : 0.8183
  9%|▊         | 26/300 [22:12<3:55:31, 51.58s/it]
Training Loss : 0.3603032011795044, Testing Loss : 0.5702375289916992,
Training Accuracy : 0.87504, Testing Accuracy : 0.8187
  9%|▉         | 27/300 [23:03<3:54:53, 51.62s/it]
Training Loss : 0.34947409618377684, Testing Loss :
0.5317544902801513, Training Accuracy : 0.87896, Testing Accuracy :
0.8247
  9%|▉         | 28/300 [23:55<3:54:05, 51.64s/it]
Training Loss : 0.333891432800293, Testing Loss : 0.5275753411293029,
Training Accuracy : 0.88484, Testing Accuracy : 0.8262
 10%|▉         | 29/300 [24:46<3:51:57, 51.36s/it]
Training Loss : 0.32331400020599366, Testing Loss :
0.5458815182209015, Training Accuracy : 0.88752, Testing Accuracy :
0.8257
 10%|█         | 30/300 [25:37<3:50:27, 51.21s/it]
```

Training Loss : 0.30859502784729004, Testing Loss : 0.5101552649021148, Training Accuracy : 0.89292, Testing Accuracy : 0.8407

10%|█ | 31/300 [26:28<3:49:19, 51.15s/it]

Training Loss : 0.297551895980835, Testing Loss : 0.49635938539505003, Training Accuracy : 0.8973, Testing Accuracy : 0.8433

11%|█ | 32/300 [27:19<3:48:09, 51.08s/it]

Training Loss : 0.2895010946273804, Testing Loss : 0.49543896255493164, Training Accuracy : 0.90004, Testing Accuracy : 0.8465

11%|█ | 33/300 [28:10<3:47:10, 51.05s/it]

Training Loss : 0.2730684278106689, Testing Loss : 0.5127138098239898, Training Accuracy : 0.90396, Testing Accuracy : 0.8434

11%|█ | 34/300 [29:01<3:46:12, 51.02s/it]

Training Loss : 0.28154671871185305, Testing Loss : 0.5090762993335723, Training Accuracy : 0.90096, Testing Accuracy : 0.8393

12%|█ | 35/300 [29:52<3:45:20, 51.02s/it]

Training Loss : 0.2636207542991638, Testing Loss : 0.5519791492700576, Training Accuracy : 0.90706, Testing Accuracy : 0.8379

12%|█ | 36/300 [30:43<3:44:31, 51.03s/it]

Training Loss : 0.2529628660583496, Testing Loss : 0.5006317177772522, Training Accuracy : 0.91002, Testing Accuracy : 0.8452

12%|█ | 37/300 [31:34<3:43:27, 50.98s/it]

Training Loss : 0.24125281942367555, Testing Loss : 0.5012982848644256, Training Accuracy : 0.9163, Testing Accuracy : 0.8479

13%|█ | 38/300 [32:24<3:42:11, 50.88s/it]

Training Loss : 0.22919247676849366, Testing Loss : 0.5005824575901031, Training Accuracy : 0.91924, Testing Accuracy : 0.848

13%|█ | 39/300 [33:15<3:40:46, 50.75s/it]

Training Loss : 0.23438435400009155, Testing Loss : 0.48977648552060127, Training Accuracy : 0.9179, Testing Accuracy : 0.8498

```
 13%|█         | 40/300 [34:05<3:39:16, 50.60s/it]
```

Training Loss : 0.22820337966918947, Testing Loss : 0.5042036104679107, Training Accuracy : 0.92088, Testing Accuracy : 0.8477

```
 14%|█         | 41/300 [34:55<3:37:50, 50.47s/it]
```

Training Loss : 0.2137829765892029, Testing Loss : 0.5088005113363266, Training Accuracy : 0.92422, Testing Accuracy : 0.85

```
 14%|█         | 42/300 [35:45<3:36:48, 50.42s/it]
```

Training Loss : 0.21980208501815796, Testing Loss : 0.520375869178772, Training Accuracy : 0.92274, Testing Accuracy : 0.8486

```
 14%|█         | 43/300 [36:35<3:35:07, 50.22s/it]
```

Training Loss : 0.20946749859809877, Testing Loss : 0.4966290003299713, Training Accuracy : 0.92624, Testing Accuracy : 0.8547

```
 15%|█         | 44/300 [37:25<3:34:11, 50.20s/it]
```

Training Loss : 0.20149297178268433, Testing Loss : 0.4952106227397919, Training Accuracy : 0.929, Testing Accuracy : 0.8534

```
 15%|█         | 45/300 [38:15<3:33:24, 50.22s/it]
```

Training Loss : 0.1874660747528076, Testing Loss : 0.5145988912701607, Training Accuracy : 0.93466, Testing Accuracy : 0.8498

```
 15%|█         | 46/300 [39:06<3:32:41, 50.24s/it]
```

Training Loss : 0.18702150880813598, Testing Loss : 0.48977670736312867, Training Accuracy : 0.93498, Testing Accuracy : 0.8598

```
 16%|█         | 47/300 [39:57<3:32:33, 50.41s/it]
```

Training Loss : 0.1756519944000244, Testing Loss : 0.5081418817758561, Training Accuracy : 0.9384, Testing Accuracy : 0.8542

```
 16%|█         | 48/300 [40:47<3:32:01, 50.48s/it]
```

Training Loss : 0.17518449808120728, Testing Loss : 0.4875015179634094, Training Accuracy : 0.93844, Testing Accuracy : 0.8612

```
 16%|█         | 49/300 [41:38<3:31:34, 50.58s/it]
```

Training Loss : 0.1647834421825409, Testing Loss : 0.5007857887268067, Training Accuracy : 0.9421, Testing Accuracy : 0.8646

```
 17%|█         | 50/300 [42:29<3:30:57, 50.63s/it]

Training Loss : 0.15192322627067567, Testing Loss :
0.5059191832661629, Training Accuracy : 0.94638, Testing Accuracy :
0.8671

 17%|█         | 51/300 [43:20<3:30:14, 50.66s/it]

Training Loss : 0.1614290971660614, Testing Loss : 0.5134654028892517,
Training Accuracy : 0.94334, Testing Accuracy : 0.8573

 17%|█         | 52/300 [44:10<3:29:17, 50.64s/it]

Training Loss : 0.1509107339000702, Testing Loss : 0.5408123473882676,
Training Accuracy : 0.94678, Testing Accuracy : 0.863

 18%|█         | 53/300 [45:01<3:28:15, 50.59s/it]

Training Loss : 0.15110589974403382, Testing Loss : 0.502655261874199,
Training Accuracy : 0.94666, Testing Accuracy : 0.864

 18%|█         | 54/300 [45:50<3:26:18, 50.32s/it]

Training Loss : 0.1424460652732849, Testing Loss : 0.5382415053844452,
Training Accuracy : 0.94972, Testing Accuracy : 0.861

 18%|█         | 55/300 [46:40<3:25:14, 50.26s/it]

Training Loss : 0.14903035126686096, Testing Loss :
0.5125958861827851, Training Accuracy : 0.94682, Testing Accuracy :
0.8617

 19%|█         | 56/300 [47:31<3:24:24, 50.27s/it]

Training Loss : 0.1354026707792282, Testing Loss : 0.5375495686769486,
Training Accuracy : 0.95222, Testing Accuracy : 0.8659

 19%|█         | 57/300 [48:21<3:23:34, 50.26s/it]

Training Loss : 0.1299040067768097, Testing Loss : 0.5072418677806855,
Training Accuracy : 0.95486, Testing Accuracy : 0.864

 19%|█         | 58/300 [49:11<3:22:50, 50.29s/it]

Training Loss : 0.123052864985466, Testing Loss : 0.5406538709163666,
Training Accuracy : 0.95708, Testing Accuracy : 0.8653

 20%|█         | 59/300 [50:02<3:22:12, 50.34s/it]

Training Loss : 0.12305952748775482, Testing Loss :
0.5219962482571602, Training Accuracy : 0.9569, Testing Accuracy :
0.8627

 20%|█         | 60/300 [50:53<3:21:53, 50.47s/it]
```

```
Training Loss : 0.12544296966552734, Testing Loss :
0.5286253190994262, Training Accuracy : 0.95576, Testing Accuracy :
0.8656

  20%|██        | 61/300 [51:44<3:21:39, 50.63s/it]

Training Loss : 0.11940653246879578, Testing Loss :
0.5317654414653779, Training Accuracy : 0.95724, Testing Accuracy :
0.8695

  21%|██        | 62/300 [52:34<3:20:46, 50.61s/it]

Training Loss : 0.11534750086784362, Testing Loss :
0.5493746733784676, Training Accuracy : 0.9595, Testing Accuracy :
0.867

  21%|██        | 63/300 [53:25<3:20:03, 50.65s/it]

Training Loss : 0.12268669157028199, Testing Loss :
0.5562727218151092, Training Accuracy : 0.95588, Testing Accuracy :
0.8672

  21%|██        | 64/300 [54:16<3:19:25, 50.70s/it]

Training Loss : 0.11739386603832244, Testing Loss :
0.5565817085266114, Training Accuracy : 0.95834, Testing Accuracy :
0.864

  22%|██        | 65/300 [55:07<3:19:05, 50.83s/it]

Training Loss : 0.10797650592803955, Testing Loss :
0.5640563380241395, Training Accuracy : 0.9615, Testing Accuracy :
0.8725

  22%|██        | 66/300 [55:57<3:17:52, 50.74s/it]

Training Loss : 0.09623980707406998, Testing Loss : 0.581335001373291,
Training Accuracy : 0.9665, Testing Accuracy : 0.8705

  22%|██        | 67/300 [56:47<3:16:05, 50.50s/it]

Training Loss : 0.10001784599542618, Testing Loss :
0.5597860471725464, Training Accuracy : 0.96398, Testing Accuracy :
0.8714

  23%|██        | 68/300 [57:38<3:15:07, 50.46s/it]

Training Loss : 0.09804072681903839, Testing Loss :
0.5887591630458832, Training Accuracy : 0.96564, Testing Accuracy :
0.8717

  23%|██        | 69/300 [58:28<3:14:02, 50.40s/it]
```

```
Training Loss : 0.10021872043132782, Testing Loss :
0.5371817745685578, Training Accuracy : 0.96488, Testing Accuracy :
0.8705

 23%|██▎       | 70/300 [59:18<3:12:53, 50.32s/it]

Training Loss : 0.0933940765953064, Testing Loss : 0.5559626825332642,
Training Accuracy : 0.96692, Testing Accuracy : 0.8701

 24%|██▎       | 71/300 [1:00:08<3:11:53, 50.28s/it]

Training Loss : 0.08848387644290924, Testing Loss :
0.5610496933460236, Training Accuracy : 0.96902, Testing Accuracy :
0.8747

 24%|██▍       | 72/300 [1:00:58<3:10:40, 50.18s/it]

Training Loss : 0.09038782977581024, Testing Loss : 0.548434619140625,
Training Accuracy : 0.96802, Testing Accuracy : 0.8733

 24%|██▍       | 73/300 [1:01:49<3:10:05, 50.24s/it]

Training Loss : 0.08742150916576386, Testing Loss :
0.5569589295864105, Training Accuracy : 0.96896, Testing Accuracy :
0.8735

 25%|██▍       | 74/300 [1:02:39<3:09:56, 50.43s/it]

Training Loss : 0.08520711573123932, Testing Loss :
0.5764333723068237, Training Accuracy : 0.969, Testing Accuracy :
0.8756

 25%|██▌       | 75/300 [1:03:30<3:09:22, 50.50s/it]

Training Loss : 0.08718063261508942, Testing Loss :
0.5357730390071869, Training Accuracy : 0.9685, Testing Accuracy :
0.8772

 25%|██▌       | 76/300 [1:04:21<3:08:51, 50.59s/it]

Training Loss : 0.07402051161527634, Testing Loss :
0.5692950594425201, Training Accuracy : 0.97336, Testing Accuracy :
0.8768

 26%|██▌       | 77/300 [1:05:12<3:08:19, 50.67s/it]

Training Loss : 0.0813014866566658, Testing Loss : 0.5482610456943512,
Training Accuracy : 0.97154, Testing Accuracy : 0.8764

 26%|██▌       | 78/300 [1:06:02<3:07:35, 50.70s/it]

Training Loss : 0.07592426122665405, Testing Loss : 0.582771567440033,
Training Accuracy : 0.97304, Testing Accuracy : 0.8717
```

26%|███ | 79/300 [1:06:53<3:06:54, 50.74s/it]

Training Loss : 0.07549251779317856, Testing Loss : 0.5756831137657166, Training Accuracy : 0.97336, Testing Accuracy : 0.8771

27%|███ | 80/300 [1:07:44<3:06:15, 50.80s/it]

Training Loss : 0.07629768316268921, Testing Loss : 0.5614856718301773, Training Accuracy : 0.9737, Testing Accuracy : 0.8762

27%|███ | 81/300 [1:08:35<3:04:54, 50.66s/it]

Training Loss : 0.07474256095886231, Testing Loss : 0.5724349199593067, Training Accuracy : 0.97372, Testing Accuracy : 0.8788

27%|███ | 82/300 [1:09:25<3:03:50, 50.60s/it]

Training Loss : 0.06729017317056656, Testing Loss : 0.5620022311210632, Training Accuracy : 0.97664, Testing Accuracy : 0.8782

28%|███ | 83/300 [1:10:15<3:02:08, 50.36s/it]

Training Loss : 0.06492812232017517, Testing Loss : 0.5829741599559783, Training Accuracy : 0.97716, Testing Accuracy : 0.8824

28%|███ | 84/300 [1:11:05<3:01:23, 50.39s/it]

Training Loss : 0.07529858441352844, Testing Loss : 0.5749461553096771, Training Accuracy : 0.97418, Testing Accuracy : 0.8746

28%|███ | 85/300 [1:11:56<3:00:32, 50.38s/it]

Training Loss : 0.06917279324293137, Testing Loss : 0.5659271295070648, Training Accuracy : 0.97596, Testing Accuracy : 0.8776

29%|███ | 86/300 [1:12:46<2:59:28, 50.32s/it]

Training Loss : 0.0637290560889244, Testing Loss : 0.6108410287857056, Training Accuracy : 0.97802, Testing Accuracy : 0.8719

29%|███ | 87/300 [1:13:36<2:58:19, 50.23s/it]

Training Loss : 0.06488212545871734, Testing Loss : 0.5884144909307361, Training Accuracy : 0.97776, Testing Accuracy : 0.8737

29%|███ | 88/300 [1:14:26<2:57:21, 50.20s/it]

Training Loss : 0.06359002823710441, Testing Loss : 0.5769033322364092, Training Accuracy : 0.97706, Testing Accuracy : 0.881

30%|███ | 89/300 [1:15:17<2:57:04, 50.35s/it]

Training Loss : 0.06000076622247696, Testing Loss : 0.5617891439914703, Training Accuracy : 0.97888, Testing Accuracy : 0.8845

30%|███ | 90/300 [1:16:07<2:56:30, 50.43s/it]

Training Loss : 0.058561078114509585, Testing Loss : 0.5760024105548859, Training Accuracy : 0.97884, Testing Accuracy : 0.8766

30%|███ | 91/300 [1:16:58<2:55:50, 50.48s/it]

Training Loss : 0.06195082460999489, Testing Loss : 0.6081012191772461, Training Accuracy : 0.9794, Testing Accuracy : 0.8792

31%|███ | 92/300 [1:17:49<2:55:42, 50.68s/it]

Training Loss : 0.0625661758697033, Testing Loss : 0.6041311296582222, Training Accuracy : 0.97788, Testing Accuracy : 0.8774

31%|███ | 93/300 [1:18:40<2:54:43, 50.65s/it]

Training Loss : 0.05998767369747162, Testing Loss : 0.5954461766242981, Training Accuracy : 0.97994, Testing Accuracy : 0.8793

31%|███ | 94/300 [1:19:30<2:54:00, 50.68s/it]

Training Loss : 0.05506057689547539, Testing Loss : 0.5667503011226654, Training Accuracy : 0.98052, Testing Accuracy : 0.8865

32%|███ | 95/300 [1:20:22<2:53:36, 50.81s/it]

Training Loss : 0.05546341473698616, Testing Loss : 0.5873045706748963, Training Accuracy : 0.98082, Testing Accuracy : 0.8802

32%|███ | 96/300 [1:21:12<2:52:29, 50.73s/it]

Training Loss : 0.052415059620141986, Testing Loss : 0.6032609618663788, Training Accuracy : 0.98164, Testing Accuracy : 0.8804

32%|███ | 97/300 [1:22:03<2:51:25, 50.67s/it]

Training Loss : 0.04958389437675476, Testing Loss : 0.6128896927833557, Training Accuracy : 0.9822, Testing Accuracy : 0.8757

33%|███ | 98/300 [1:22:53<2:50:18, 50.59s/it]

Training Loss : 0.0512255098426342, Testing Loss : 0.6507545437812805, Training Accuracy : 0.9819, Testing Accuracy : 0.879

33%|███ | 99/300 [1:23:43<2:49:13, 50.51s/it]

Training Loss : 0.04946521449804306, Testing Loss : 0.6517802816390991, Training Accuracy : 0.98274, Testing Accuracy : 0.8811

33%|███ | 100/300 [1:24:34<2:48:28, 50.54s/it]

Training Loss : 0.048657758920192716, Testing Loss : 0.6324366926670074, Training Accuracy : 0.98282, Testing Accuracy : 0.8838

34%|███ | 101/300 [1:25:24<2:47:18, 50.45s/it]

Training Loss : 0.05413148109912872, Testing Loss : 0.6033781976461411, Training Accuracy : 0.98136, Testing Accuracy : 0.8803

34%|███ | 102/300 [1:26:14<2:46:14, 50.38s/it]

Training Loss : 0.04958187876582146, Testing Loss : 0.6471959753990173, Training Accuracy : 0.98196, Testing Accuracy : 0.8833

34%|███ | 103/300 [1:27:05<2:45:20, 50.36s/it]

Training Loss : 0.04618874662458897, Testing Loss : 0.5995940048217774, Training Accuracy : 0.98422, Testing Accuracy : 0.8856

35%|███ | 104/300 [1:27:55<2:44:41, 50.41s/it]

Training Loss : 0.04736058033943176, Testing Loss : 0.6355292096138, Training Accuracy : 0.98228, Testing Accuracy : 0.881

35%|███ | 105/300 [1:28:46<2:44:22, 50.58s/it]

Training Loss : 0.04387015375316143, Testing Loss : 0.6181062396049499, Training Accuracy : 0.98494, Testing Accuracy : 0.8855

35%|███ | 106/300 [1:29:37<2:43:34, 50.59s/it]

Training Loss : 0.04656225181460381, Testing Loss : 0.6525047996759414, Training Accuracy : 0.9834, Testing Accuracy : 0.8798

 36%|███▌ | 107/300 [1:30:27<2:42:45, 50.60s/it]

Training Loss : 0.04644400257885456, Testing Loss : 0.6459421191215515, Training Accuracy : 0.9842, Testing Accuracy : 0.8803

 36%|███▌ | 108/300 [1:31:18<2:42:02, 50.64s/it]

Training Loss : 0.04495429239869118, Testing Loss : 0.6364593900203704, Training Accuracy : 0.98496, Testing Accuracy : 0.8796

 36%|███▋ | 109/300 [1:32:09<2:41:25, 50.71s/it]

Training Loss : 0.04425027590036392, Testing Loss : 0.5945832233428955, Training Accuracy : 0.98438, Testing Accuracy : 0.8854

 37%|███▋ | 110/300 [1:33:00<2:40:42, 50.75s/it]

Training Loss : 0.037504522157907484, Testing Loss : 0.6262766248703003, Training Accuracy : 0.98664, Testing Accuracy : 0.8867

 37%|███▋ | 111/300 [1:33:51<2:39:52, 50.75s/it]

Training Loss : 0.03700010726749897, Testing Loss : 0.6404787611007691, Training Accuracy : 0.9872, Testing Accuracy : 0.8853

 37%|███▋ | 112/300 [1:34:41<2:38:58, 50.74s/it]

Training Loss : 0.042221009444594386, Testing Loss : 0.6423442483901978, Training Accuracy : 0.98526, Testing Accuracy : 0.8866

 38%|███▊ | 113/300 [1:35:32<2:38:07, 50.74s/it]

Training Loss : 0.043882696566581725, Testing Loss : 0.6721553740501404, Training Accuracy : 0.98508, Testing Accuracy : 0.8783

 38%|███▊ | 114/300 [1:36:23<2:37:10, 50.70s/it]

Training Loss : 0.04315584982156753, Testing Loss : 0.6256549303054809, Training Accuracy : 0.98504, Testing Accuracy : 0.8826

 38%|███▊ | 115/300 [1:37:12<2:35:24, 50.40s/it]

```
Training Loss : 0.040865358310937884, Testing Loss :
0.6592369837760925, Training Accuracy : 0.9858, Testing Accuracy :
0.8822

 39%|████      | 116/300 [1:38:03<2:34:25, 50.35s/it]

Training Loss : 0.03721206472814083, Testing Loss :
0.6277361439704895, Training Accuracy : 0.98734, Testing Accuracy :
0.8835

 39%|████      | 117/300 [1:38:53<2:33:45, 50.42s/it]

Training Loss : 0.03624783056288958, Testing Loss :
0.6146328484535217, Training Accuracy : 0.98748, Testing Accuracy :
0.8869

 39%|████      | 118/300 [1:39:44<2:33:16, 50.53s/it]

Training Loss : 0.036178605889976026, Testing Loss :
0.6154830747246742, Training Accuracy : 0.98788, Testing Accuracy :
0.8855

 40%|████      | 119/300 [1:40:35<2:32:26, 50.53s/it]

Training Loss : 0.035617525699436665, Testing Loss :
0.6841199934959412, Training Accuracy : 0.98844, Testing Accuracy :
0.8791

 40%|████      | 120/300 [1:41:25<2:31:27, 50.49s/it]

Training Loss : 0.03496115669608116, Testing Loss :
0.6675833118438721, Training Accuracy : 0.98848, Testing Accuracy :
0.8806

 40%|████      | 121/300 [1:42:15<2:30:23, 50.41s/it]

Training Loss : 0.03739699281305075, Testing Loss : 0.659401105594635,
Training Accuracy : 0.98762, Testing Accuracy : 0.8827

 41%|████      | 122/300 [1:43:06<2:29:51, 50.52s/it]

Training Loss : 0.03177539518147707, Testing Loss :
0.6374552897453308, Training Accuracy : 0.98888, Testing Accuracy :
0.8861

 41%|████      | 123/300 [1:43:57<2:29:17, 50.61s/it]

Training Loss : 0.031132422499656677, Testing Loss :
0.6537699572563171, Training Accuracy : 0.98958, Testing Accuracy :
0.8862

 41%|████      | 124/300 [1:44:48<2:28:58, 50.78s/it]
```

Training Loss : 0.03041423185825348, Testing Loss : 0.6939976770401001, Training Accuracy : 0.98924, Testing Accuracy : 0.885

42%|████ | 125/300 [1:45:39<2:28:18, 50.85s/it]

Training Loss : 0.03102758801341057, Testing Loss : 0.680868739938736, Training Accuracy : 0.98898, Testing Accuracy : 0.8879

42%|████ | 126/300 [1:46:30<2:27:41, 50.93s/it]

Training Loss : 0.029260787582919, Testing Loss : 0.6545189796328544, Training Accuracy : 0.9899, Testing Accuracy : 0.8868

42%|████ | 127/300 [1:47:21<2:27:05, 51.01s/it]

Training Loss : 0.027937346759438514, Testing Loss : 0.6815939101815224, Training Accuracy : 0.99058, Testing Accuracy : 0.8898

43%|████ | 128/300 [1:48:12<2:26:08, 50.98s/it]

Training Loss : 0.02808529759436846, Testing Loss : 0.6505299057483673, Training Accuracy : 0.99008, Testing Accuracy : 0.8881

43%|████ | 129/300 [1:49:03<2:24:54, 50.84s/it]

Training Loss : 0.025972129683494567, Testing Loss : 0.639175733691454, Training Accuracy : 0.99118, Testing Accuracy : 0.891

43%|████ | 130/300 [1:49:53<2:23:36, 50.69s/it]

Training Loss : 0.026400906725525856, Testing Loss : 0.6511851387500763, Training Accuracy : 0.99036, Testing Accuracy : 0.8914

44%|████ | 131/300 [1:50:43<2:22:27, 50.57s/it]

Training Loss : 0.022885717538148165, Testing Loss : 0.6349695517063141, Training Accuracy : 0.99264, Testing Accuracy : 0.8956

44%|████ | 132/300 [1:51:34<2:21:22, 50.49s/it]

Training Loss : 0.025500244659930466, Testing Loss : 0.6725020387470723, Training Accuracy : 0.99106, Testing Accuracy : 0.8892

44%|████ | 133/300 [1:52:24<2:20:30, 50.48s/it]

Training Loss : 0.027893756040632725, Testing Loss : 0.7095812375545502, Training Accuracy : 0.9905, Testing Accuracy : 0.8838

 45%|████▌ | 134/300 [1:53:14<2:19:30, 50.42s/it]

Training Loss : 0.027228274478316307, Testing Loss : 0.634157137261331, Training Accuracy : 0.9904, Testing Accuracy : 0.8897

 45%|████▌ | 135/300 [1:54:05<2:18:50, 50.49s/it]

Training Loss : 0.030308986651003362, Testing Loss : 0.653934158372879, Training Accuracy : 0.98966, Testing Accuracy : 0.892

 45%|████▌ | 136/300 [1:54:55<2:17:41, 50.37s/it]

Training Loss : 0.024809992222487927, Testing Loss : 0.6362481813907623, Training Accuracy : 0.99106, Testing Accuracy : 0.8896

 46%|████▌ | 137/300 [1:55:45<2:16:44, 50.34s/it]

Training Loss : 0.022254395698830486, Testing Loss : 0.6826085461616516, Training Accuracy : 0.99242, Testing Accuracy : 0.8882

 46%|████▌ | 138/300 [1:56:36<2:16:04, 50.40s/it]

Training Loss : 0.025402451038211583, Testing Loss : 0.6885506557703018, Training Accuracy : 0.99128, Testing Accuracy : 0.8869

 46%|████▋ | 139/300 [1:57:26<2:15:21, 50.44s/it]

Training Loss : 0.02947509616240859, Testing Loss : 0.6615972246170044, Training Accuracy : 0.99038, Testing Accuracy : 0.8917

 47%|████▋ | 140/300 [1:58:17<2:14:40, 50.50s/it]

Training Loss : 0.026415507250279188, Testing Loss : 0.6386092526435853, Training Accuracy : 0.99128, Testing Accuracy : 0.8921

 47%|████▋ | 141/300 [1:59:08<2:14:06, 50.61s/it]

Training Loss : 0.024208767513334752, Testing Loss : 0.6778791234970093, Training Accuracy : 0.99126, Testing Accuracy : 0.8903

 47%|████▋ | 142/300 [1:59:59<2:13:31, 50.70s/it]

Training Loss : 0.02326321456357837, Testing Loss : 0.6820631877422333, Training Accuracy : 0.99144, Testing Accuracy : 0.8894

48%|████▊ | 143/300 [2:00:50<2:12:46, 50.74s/it]

Training Loss : 0.023271476458385586, Testing Loss : 0.6661824361801147, Training Accuracy : 0.99134, Testing Accuracy : 0.8898

48%|████▊ | 144/300 [2:01:41<2:12:00, 50.77s/it]

Training Loss : 0.023765823556035758, Testing Loss : 0.6415060368061065, Training Accuracy : 0.9915, Testing Accuracy : 0.8907

48%|████▊ | 145/300 [2:02:31<2:10:45, 50.62s/it]

Training Loss : 0.02255005111724138, Testing Loss : 0.6534097477912902, Training Accuracy : 0.99228, Testing Accuracy : 0.8926

49%|████▊ | 146/300 [2:03:21<2:09:19, 50.39s/it]

Training Loss : 0.022334132117033004, Testing Loss : 0.6628198947906494, Training Accuracy : 0.99234, Testing Accuracy : 0.8894

49%|████▉ | 147/300 [2:04:11<2:08:13, 50.28s/it]

Training Loss : 0.025406835048496723, Testing Loss : 0.6226226906299591, Training Accuracy : 0.9914, Testing Accuracy : 0.8931

49%|████▉ | 148/300 [2:05:01<2:07:06, 50.17s/it]

Training Loss : 0.01857137003861368, Testing Loss : 0.6876132871687413, Training Accuracy : 0.99344, Testing Accuracy : 0.8906

50%|████▉ | 149/300 [2:05:50<2:06:01, 50.07s/it]

Training Loss : 0.0205702114585042, Testing Loss : 0.659669110298156 8,
Training Accuracy : 0.99284, Testing Accuracy : 0.894

50%|█████ | 150/300 [2:06:41<2:05:16, 50.11s/it]

Training Loss : 0.01997320263840258, Testing Loss : 0.7110150890827179, Training Accuracy : 0.99328, Testing Accuracy : 0.8916

50%|█████ | 151/300 [2:07:31<2:04:33, 50.15s/it]

Training Loss : 0.021204572284519673, Testing Loss : 0.6970590429902077, Training Accuracy : 0.99274, Testing Accuracy : 0.8899

51%|█████ | 152/300 [2:08:21<2:03:56, 50.25s/it]

Training Loss : 0.022708670855164526, Testing Loss : 0.6964296731710434, Training Accuracy : 0.99198, Testing Accuracy : 0.8906

51%|█████ | 153/300 [2:09:12<2:03:22, 50.36s/it]

Training Loss : 0.019670491465330123, Testing Loss : 0.6812108939170838, Training Accuracy : 0.99312, Testing Accuracy : 0.8922

51%|█████ | 154/300 [2:10:03<2:02:53, 50.50s/it]

Training Loss : 0.019691694493480027, Testing Loss : 0.6838801940202713, Training Accuracy : 0.9932, Testing Accuracy : 0.8932

52%|█████ | 155/300 [2:10:53<2:02:04, 50.52s/it]

Training Loss : 0.019691645381897688, Testing Loss : 0.6583851360082627, Training Accuracy : 0.99328, Testing Accuracy : 0.8966

52%|█████ | 156/300 [2:11:44<2:01:20, 50.56s/it]

Training Loss : 0.01900144662104547, Testing Loss : 0.6989805149555206, Training Accuracy : 0.9933, Testing Accuracy : 0.8914

52%|█████ | 157/300 [2:12:34<2:00:25, 50.53s/it]

Training Loss : 0.019065141580030323, Testing Loss : 0.6553405778884888, Training Accuracy : 0.99362, Testing Accuracy : 0.8944

53%|█████ | 158/300 [2:13:25<1:59:28, 50.48s/it]

Training Loss : 0.017443676094934343, Testing Loss : 0.6989306135058403, Training Accuracy : 0.99416, Testing Accuracy : 0.8898

53%|█████ | 159/300 [2:14:15<1:58:09, 50.28s/it]

Training Loss : 0.01686713241806254, Testing Loss : 0.6880358777999878, Training Accuracy : 0.99454, Testing Accuracy : 0.8917

53%|█████ | 160/300 [2:15:05<1:57:02, 50.16s/it]

Training Loss : 0.018003944538384675, Testing Loss : 0.6992039823412896, Training Accuracy : 0.99358, Testing Accuracy : 0.8954

 54%|█████ | 161/300 [2:15:55<1:56:18, 50.21s/it]

Training Loss : 0.015735763298049567, Testing Loss : 0.6512037988185883, Training Accuracy : 0.9944, Testing Accuracy : 0.8966

 54%|█████ | 162/300 [2:16:45<1:55:35, 50.26s/it]

Training Loss : 0.0122647372803092, Testing Loss : 0.6690608282566071, Training Accuracy : 0.99574, Testing Accuracy : 0.9023

 54%|█████ | 163/300 [2:17:36<1:54:53, 50.32s/it]

Training Loss : 0.013052098441496492, Testing Loss : 0.6617214141368866, Training Accuracy : 0.99556, Testing Accuracy : 0.8978

 55%|█████ | 164/300 [2:18:26<1:54:00, 50.30s/it]

Training Loss : 0.01475037628505379, Testing Loss : 0.7328686270713806, Training Accuracy : 0.99506, Testing Accuracy : 0.8903

 55%|█████ | 165/300 [2:19:16<1:53:08, 50.28s/it]

Training Loss : 0.014488786079920829, Testing Loss : 0.7260583797454834, Training Accuracy : 0.9948, Testing Accuracy : 0.8919

 55%|█████ | 166/300 [2:20:07<1:52:31, 50.38s/it]

Training Loss : 0.015977208711206912, Testing Loss : 0.7055716282367707, Training Accuracy : 0.99462, Testing Accuracy : 0.8936

 56%|█████ | 167/300 [2:20:58<1:52:12, 50.62s/it]

Training Loss : 0.015117824024260044, Testing Loss : 0.7070741997599602, Training Accuracy : 0.99484, Testing Accuracy : 0.895

 56%|█████ | 168/300 [2:21:49<1:51:30, 50.69s/it]

Training Loss : 0.01466093176290393, Testing Loss : 0.6993438965439797, Training Accuracy : 0.99496, Testing Accuracy : 0.8946

 56%|█████ | 169/300 [2:22:39<1:50:37, 50.67s/it]

```
Training Loss : 0.013977533716969193, Testing Loss :
0.7077382449150086, Training Accuracy : 0.99504, Testing Accuracy :
0.8957

  57%|█████▋    | 170/300 [2:23:30<1:49:46, 50.66s/it]

Training Loss : 0.015194293287023902, Testing Loss :
0.7538093051910401, Training Accuracy : 0.99488, Testing Accuracy :
0.8901

  57%|█████▋    | 171/300 [2:24:21<1:48:53, 50.65s/it]

Training Loss : 0.01562893845655024, Testing Loss :
0.6973407278060914, Training Accuracy : 0.9946, Testing Accuracy :
0.896

  57%|█████▋    | 172/300 [2:25:11<1:48:06, 50.68s/it]

Training Loss : 0.01373029165148735, Testing Loss :
0.7435402807235718, Training Accuracy : 0.99534, Testing Accuracy :
0.8904

  58%|█████▊    | 173/300 [2:26:02<1:46:56, 50.53s/it]

Training Loss : 0.016721475045569242, Testing Loss :
0.6597676332473755, Training Accuracy : 0.99444, Testing Accuracy :
0.8973

  58%|█████▊    | 174/300 [2:26:52<1:46:03, 50.50s/it]

Training Loss : 0.015466195471659302, Testing Loss :
0.7163176023483276, Training Accuracy : 0.99476, Testing Accuracy :
0.8931

  58%|█████▊    | 175/300 [2:27:42<1:44:55, 50.36s/it]

Training Loss : 0.01462813229098916, Testing Loss : 0.734309476184845,
Training Accuracy : 0.99516, Testing Accuracy : 0.8884

  59%|█████▉    | 176/300 [2:28:32<1:43:46, 50.21s/it]

Training Loss : 0.014668660136014222, Testing Loss : 0.70796555352211,
Training Accuracy : 0.99522, Testing Accuracy : 0.8935

  59%|█████▉    | 177/300 [2:29:22<1:42:52, 50.18s/it]

Training Loss : 0.01202560352526605, Testing Loss :
0.6776410574197769, Training Accuracy : 0.99596, Testing Accuracy :
0.8988

  59%|█████▉    | 178/300 [2:30:12<1:42:00, 50.17s/it]
```

Training Loss : 0.011218566990830004, Testing Loss : 0.6903307695388794, Training Accuracy : 0.99636, Testing Accuracy : 0.8969

60%|██████ | 179/300 [2:31:02<1:41:00, 50.09s/it]

Training Loss : 0.013758614414259792, Testing Loss : 0.7195565972805024, Training Accuracy : 0.99542, Testing Accuracy : 0.8922

60%|██████ | 180/300 [2:31:52<1:40:19, 50.16s/it]

Training Loss : 0.012690397947467864, Testing Loss : 0.668628975391388, Training Accuracy : 0.99564, Testing Accuracy : 0.8973

60%|██████ | 181/300 [2:32:42<1:39:24, 50.12s/it]

Training Loss : 0.011131021630465985, Testing Loss : 0.7177944608211517, Training Accuracy : 0.99602, Testing Accuracy : 0.8932

61%|██████ | 182/300 [2:33:33<1:38:48, 50.25s/it]

Training Loss : 0.00994527231303975, Testing Loss : 0.7333163456439972, Training Accuracy : 0.99648, Testing Accuracy : 0.893

61%|██████ | 183/300 [2:34:24<1:38:21, 50.44s/it]

Training Loss : 0.01060296529341489, Testing Loss : 0.7255939009748399, Training Accuracy : 0.99646, Testing Accuracy : 0.8984

61%|██████▏ | 184/300 [2:35:15<1:37:45, 50.57s/it]

Training Loss : 0.009633948159497231, Testing Loss : 0.7208732516527175, Training Accuracy : 0.99672, Testing Accuracy : 0.8998

62%|██████▏ | 185/300 [2:36:06<1:37:08, 50.69s/it]

Training Loss : 0.008420184018146247, Testing Loss : 0.7365788826584816, Training Accuracy : 0.997, Testing Accuracy : 0.8993

62%|██████▏ | 186/300 [2:36:56<1:36:15, 50.66s/it]

Training Loss : 0.009205446767769753, Testing Loss : 0.7004448023557663, Training Accuracy : 0.99712, Testing Accuracy : 0.8985

62%|██████▏ | 187/300 [2:37:47<1:35:31, 50.72s/it]

Training Loss : 0.009757723063994198, Testing Loss : 0.7403268857955932, Training Accuracy : 0.99666, Testing Accuracy : 0.8986

63%|██████ | 188/300 [2:38:38<1:34:38, 50.70s/it]

Training Loss : 0.00878474524602294, Testing Loss : 0.7006562445700169, Training Accuracy : 0.99692, Testing Accuracy : 0.9003

63%|██████ | 189/300 [2:39:28<1:33:38, 50.62s/it]

Training Loss : 0.008622655976545065, Testing Loss : 0.7350100099951029, Training Accuracy : 0.99714, Testing Accuracy : 0.9007

63%|██████ | 190/300 [2:40:18<1:32:29, 50.45s/it]

Training Loss : 0.00830190166471526, Testing Loss : 0.7119323916921392, Training Accuracy : 0.9971, Testing Accuracy : 0.901

64%|██████ | 191/300 [2:41:09<1:31:40, 50.47s/it]

Training Loss : 0.008356954285241663, Testing Loss : 0.74498978587389, Training Accuracy : 0.99718, Testing Accuracy : 0.8989

64%|██████ | 192/300 [2:41:59<1:30:32, 50.30s/it]

Training Loss : 0.007590013332497329, Testing Loss : 0.7306645954981447, Training Accuracy : 0.99754, Testing Accuracy : 0.8993

64%|██████ | 193/300 [2:42:49<1:29:51, 50.39s/it]

Training Loss : 0.0077568415182176975, Testing Loss : 0.7362108430922032, Training Accuracy : 0.9975, Testing Accuracy : 0.9001

65%|██████ | 194/300 [2:43:40<1:28:54, 50.33s/it]

Training Loss : 0.006224604171682149, Testing Loss : 0.6966277525067329, Training Accuracy : 0.99792, Testing Accuracy : 0.9056

65%|██████ | 195/300 [2:44:30<1:28:00, 50.29s/it]

Training Loss : 0.006820096943560057, Testing Loss : 0.7449097651228309, Training Accuracy : 0.99778, Testing Accuracy : 0.8993

65%|██████ | 196/300 [2:45:20<1:27:06, 50.26s/it]

Training Loss : 0.0075996248852158894, Testing Loss : 0.7366296968936921, Training Accuracy : 0.99748, Testing Accuracy : 0.9024

66%|██████ | 197/300 [2:46:10<1:26:19, 50.29s/it]

Training Loss : 0.007214325210051611, Testing Loss : 0.7027962957382202, Training Accuracy : 0.99786, Testing Accuracy : 0.9037

66%|██████ | 198/300 [2:47:01<1:25:41, 50.41s/it]

Training Loss : 0.00834622794301249, Testing Loss : 0.7574301226779818, Training Accuracy : 0.99722, Testing Accuracy : 0.9006

66%|██████ | 199/300 [2:47:52<1:25:01, 50.51s/it]

Training Loss : 0.009599390794876962, Testing Loss : 0.7459056294403971, Training Accuracy : 0.99698, Testing Accuracy : 0.9004

67%|██████ | 200/300 [2:48:43<1:24:25, 50.66s/it]

Training Loss : 0.008272954897880554, Testing Loss : 0.720702743268013, Training Accuracy : 0.99736, Testing Accuracy : 0.899

67%|██████ | 201/300 [2:49:34<1:23:44, 50.76s/it]

Training Loss : 0.007734282403010875, Testing Loss : 0.7086509774923324, Training Accuracy : 0.99728, Testing Accuracy : 0.9037

67%|██████ | 202/300 [2:50:24<1:22:51, 50.73s/it]

Training Loss : 0.006486635559750721, Testing Loss : 0.7132209315240383, Training Accuracy : 0.99766, Testing Accuracy : 0.8987

68%|██████ | 203/300 [2:51:15<1:21:58, 50.71s/it]

Training Loss : 0.006516806951584295, Testing Loss : 0.7345041453719139, Training Accuracy : 0.99774, Testing Accuracy : 0.9052

68%|██████ | 204/300 [2:52:06<1:21:07, 50.71s/it]

Training Loss : 0.005970868375408463, Testing Loss : 0.7312974956743419, Training Accuracy : 0.99796, Testing Accuracy : 0.8992

68%|██████ | 205/300 [2:52:56<1:20:08, 50.62s/it]

Training Loss : 0.006680877727754414, Testing Loss : 0.7300891924858093, Training Accuracy : 0.99782, Testing Accuracy : 0.9034

 69%|███████ | 206/300 [2:53:47<1:19:20, 50.64s/it]

Training Loss : 0.005604279637532308, Testing Loss : 0.7622229617458536, Training Accuracy : 0.99804, Testing Accuracy : 0.9014

 69%|███████ | 207/300 [2:54:37<1:18:26, 50.61s/it]

Training Loss : 0.006323175489502028, Testing Loss : 0.7770417173637078, Training Accuracy : 0.99778, Testing Accuracy : 0.9003

 69%|███████ | 208/300 [2:55:28<1:17:34, 50.60s/it]

Training Loss : 0.007918860203605145, Testing Loss : 0.7798761849403382, Training Accuracy : 0.99732, Testing Accuracy : 0.8978

 70%|███████ | 209/300 [2:56:18<1:16:42, 50.58s/it]

Training Loss : 0.006353147006202489, Testing Loss : 0.7300230010271073, Training Accuracy : 0.99782, Testing Accuracy : 0.905

 70%|███████ | 210/300 [2:57:09<1:15:47, 50.53s/it]

Training Loss : 0.004665088733900385, Testing Loss : 0.7654248338144273, Training Accuracy : 0.9985, Testing Accuracy : 0.9

 70%|███████ | 211/300 [2:57:59<1:14:54, 50.50s/it]

Training Loss : 0.00602553690594621, Testing Loss : 0.7453021382972598, Training Accuracy : 0.99798, Testing Accuracy : 0.9046

 71%|███████ | 212/300 [2:58:50<1:14:00, 50.46s/it]

Training Loss : 0.00568577073068358, Testing Loss : 0.7650113232895732, Training Accuracy : 0.99796, Testing Accuracy : 0.9014

 71%|███████ | 213/300 [2:59:40<1:13:11, 50.48s/it]

Training Loss : 0.0035757197552500294, Testing Loss : 0.7703757662773132, Training Accuracy : 0.99876, Testing Accuracy : 0.9049

 71%|███████ | 214/300 [3:00:31<1:12:23, 50.51s/it]

Training Loss : 0.003905311906489078, Testing Loss : 0.7469755942314863, Training Accuracy : 0.99866, Testing Accuracy : 0.9046

 72%|███████▏ | 215/300 [3:01:22<1:11:42, 50.61s/it]

Training Loss : 0.004235129612458404, Testing Loss : 0.7840943446386373, Training Accuracy : 0.99876, Testing Accuracy : 0.9054

 72%|███████▏ | 216/300 [3:02:13<1:10:58, 50.70s/it]

Training Loss : 0.0045740750234993174, Testing Loss : 0.7386875282671302, Training Accuracy : 0.99858, Testing Accuracy : 0.9094

 72%|███████▏ | 217/300 [3:03:04<1:10:15, 50.79s/it]

Training Loss : 0.003753253328166902, Testing Loss : 0.7752503308534622, Training Accuracy : 0.99892, Testing Accuracy : 0.9059

 73%|███████▎ | 218/300 [3:03:54<1:09:25, 50.79s/it]

Training Loss : 0.004355270182173699, Testing Loss : 0.7598964664936065, Training Accuracy : 0.99862, Testing Accuracy : 0.9056

 73%|███████▎ | 219/300 [3:04:45<1:08:38, 50.84s/it]

Training Loss : 0.004397548911147751, Testing Loss : 0.795757759880647, Training Accuracy : 0.99842, Testing Accuracy : 0.9035

 73%|███████▎ | 220/300 [3:05:36<1:07:47, 50.84s/it]

Training Loss : 0.003686723817954771, Testing Loss : 0.7780941040992737, Training Accuracy : 0.99866, Testing Accuracy : 0.9062

 74%|███████▎ | 221/300 [3:06:27<1:06:55, 50.83s/it]

Training Loss : 0.0033622452633827924, Testing Loss : 0.7768267453793436, Training Accuracy : 0.99886, Testing Accuracy : 0.9029

 74%|███████▍ | 222/300 [3:07:18<1:06:02, 50.81s/it]

Training Loss : 0.003506753009986132, Testing Loss : 0.7912163743019104, Training Accuracy : 0.99864, Testing Accuracy : 0.9047

 74%|███████▍ | 223/300 [3:08:08<1:05:08, 50.76s/it]

```
Training Loss : 0.003620079993578838, Testing Loss :
0.7949525101661682, Training Accuracy : 0.9988, Testing Accuracy :
0.9016

  75%|███████▍  | 224/300 [3:08:58<1:04:01, 50.54s/it]

Training Loss : 0.003473418298056349, Testing Loss :
0.7913781638145447, Training Accuracy : 0.99882, Testing Accuracy :
0.9046

  75%|███████▌  | 225/300 [3:09:49<1:03:07, 50.50s/it]

Training Loss : 0.0029849347038939595, Testing Loss :
0.7910072970390319, Training Accuracy : 0.99898, Testing Accuracy :
0.9032

  75%|███████▌  | 226/300 [3:10:39<1:02:11, 50.43s/it]

Training Loss : 0.0036314157366892324, Testing Loss :
0.7846135242938995, Training Accuracy : 0.99878, Testing Accuracy :
0.9053

  76%|███████▌  | 227/300 [3:11:30<1:01:24, 50.47s/it]

Training Loss : 0.004939615928898565, Testing Loss :
0.8132630966603756, Training Accuracy : 0.99828, Testing Accuracy :
0.9018

  76%|███████▌  | 228/300 [3:12:20<1:00:31, 50.44s/it]

Training Loss : 0.004676851911157137, Testing Loss :
0.7964630191326141, Training Accuracy : 0.99854, Testing Accuracy :
0.902

  76%|███████▋  | 229/300 [3:13:10<59:37, 50.39s/it]

Training Loss : 0.003741807332027238, Testing Loss :
0.8093729397031478, Training Accuracy : 0.99872, Testing Accuracy :
0.9024

  77%|███████▋  | 230/300 [3:14:01<58:46, 50.38s/it]

Training Loss : 0.003531383112471085, Testing Loss :
0.7580372143745422, Training Accuracy : 0.99874, Testing Accuracy :
0.9071

  77%|███████▋  | 231/300 [3:14:51<58:01, 50.46s/it]

Training Loss : 0.002879816721929237, Testing Loss :
0.8007704183644615, Training Accuracy : 0.99912, Testing Accuracy :
0.9023

  77%|███████▋  | 232/300 [3:15:42<57:21, 50.61s/it]
```

Training Loss : 0.004246280844022986, Testing Loss : 0.7689806571960449, Training Accuracy : 0.99868, Testing Accuracy : 0.9052

78%|███████▊ | 233/300 [3:16:33<56:38, 50.72s/it]

Training Loss : 0.003737315776066389, Testing Loss : 0.7962198976039887, Training Accuracy : 0.9987, Testing Accuracy : 0.904

78%|███████▊ | 234/300 [3:17:24<55:51, 50.79s/it]

Training Loss : 0.003306269658000674, Testing Loss : 0.7969719643592834, Training Accuracy : 0.99888, Testing Accuracy : 0.9045

78%|███████▊ | 235/300 [3:18:15<55:03, 50.82s/it]

Training Loss : 0.0028271913685253823, Testing Loss : 0.8261069929096848, Training Accuracy : 0.99902, Testing Accuracy : 0.902

79%|███████▊ | 236/300 [3:19:06<54:08, 50.76s/it]

Training Loss : 0.0033495964770019052, Testing Loss : 0.783774348282814, Training Accuracy : 0.99878, Testing Accuracy : 0.9062

79%|███████▉ | 237/300 [3:19:56<53:18, 50.78s/it]

Training Loss : 0.004685987336416729, Testing Loss : 0.8202160791006318, Training Accuracy : 0.99842, Testing Accuracy : 0.9002

79%|███████▉ | 238/300 [3:20:47<52:28, 50.79s/it]

Training Loss : 0.004844427455810364, Testing Loss : 0.7780877282619476, Training Accuracy : 0.9983, Testing Accuracy : 0.9072

80%|███████▉ | 239/300 [3:21:38<51:30, 50.67s/it]

Training Loss : 0.002890996442614123, Testing Loss : 0.8030858982939971, Training Accuracy : 0.99898, Testing Accuracy : 0.9033

80%|████████ | 240/300 [3:22:28<50:37, 50.62s/it]

Training Loss : 0.0038544360631587917, Testing Loss : 0.7792401895999909, Training Accuracy : 0.99882, Testing Accuracy : 0.9058

80%|████████ | 241/300 [3:23:18<49:38, 50.48s/it]

Training Loss : 0.0029335718064094542, Testing Loss : 0.76960290924547, Training Accuracy : 0.999, Testing Accuracy : 0.9046

81%|████████ | 242/300 [3:24:09<48:44, 50.42s/it]

Training Loss : 0.004098975933741312, Testing Loss : 0.7652599495887756, Training Accuracy : 0.99868, Testing Accuracy : 0.9081

81%|████████ | 243/300 [3:24:59<47:54, 50.43s/it]

Training Loss : 0.002986559513923712, Testing Loss : 0.7666340408682824, Training Accuracy : 0.99898, Testing Accuracy : 0.9066

81%|████████▏ | 244/300 [3:25:49<46:58, 50.34s/it]

Training Loss : 0.0025886223200405948, Testing Loss : 0.7732638213157654, Training Accuracy : 0.99914, Testing Accuracy : 0.9077

82%|████████▏ | 245/300 [3:26:39<46:05, 50.28s/it]

Training Loss : 0.002509620467007626, Testing Loss : 0.7765979819059372, Training Accuracy : 0.99922, Testing Accuracy : 0.9054

82%|████████▏ | 246/300 [3:27:29<45:09, 50.18s/it]

Training Loss : 0.0019124393553775734, Testing Loss : 0.7851211951971054, Training Accuracy : 0.9994, Testing Accuracy : 0.9087

82%|████████▏ | 247/300 [3:28:20<44:21, 50.21s/it]

Training Loss : 0.0017167088804626838, Testing Loss : 0.7436692133426667, Training Accuracy : 0.99946, Testing Accuracy : 0.9071

83%|████████▎ | 248/300 [3:29:10<43:31, 50.23s/it]

Training Loss : 0.001855282965627266, Testing Loss : 0.7800235723853112, Training Accuracy : 0.99938, Testing Accuracy : 0.9088

83%|████████▎ | 249/300 [3:30:01<42:51, 50.43s/it]

Training Loss : 0.0021939013425959276, Testing Loss : 0.7723087565898895, Training Accuracy : 0.99934, Testing Accuracy : 0.9064

83%|████████▎ | 250/300 [3:30:52<42:09, 50.59s/it]

Training Loss : 0.002229254486135906, Testing Loss : 0.7938468881607056, Training Accuracy : 0.99922, Testing Accuracy : 0.9071

 84%|████████ | 251/300 [3:31:43<41:23, 50.69s/it]

Training Loss : 0.0016865817643213086, Testing Loss : 0.7882385025024414, Training Accuracy : 0.9995, Testing Accuracy : 0.909

 84%|████████ | 252/300 [3:32:35<40:54, 51.13s/it]

Training Loss : 0.0020416742217564026, Testing Loss : 0.7759445066452026, Training Accuracy : 0.99928, Testing Accuracy : 0.9071

 84%|████████ | 253/300 [3:33:27<40:15, 51.40s/it]

Training Loss : 0.0018989222688565496, Testing Loss : 0.7955179936885833, Training Accuracy : 0.99932, Testing Accuracy : 0.9093

 85%|████████ | 254/300 [3:34:18<39:22, 51.37s/it]

Training Loss : 0.0019302759303583298, Testing Loss : 0.7950831510543823, Training Accuracy : 0.99932, Testing Accuracy : 0.9064

 85%|████████ | 255/300 [3:35:10<38:33, 51.41s/it]

Training Loss : 0.0020588851689663716, Testing Loss : 0.7866692175269127, Training Accuracy : 0.99936, Testing Accuracy : 0.9084

 85%|████████ | 256/300 [3:36:01<37:43, 51.44s/it]

Training Loss : 0.0015654064272413962, Testing Loss : 0.7776320552527904, Training Accuracy : 0.9995, Testing Accuracy : 0.9077

 86%|████████ | 257/300 [3:36:53<36:56, 51.55s/it]

Training Loss : 0.0011326909680047538, Testing Loss : 0.7898563123185187, Training Accuracy : 0.99966, Testing Accuracy : 0.9105

 86%|████████ | 258/300 [3:37:45<36:06, 51.58s/it]

Training Loss : 0.0009324537622410571, Testing Loss : 0.7838878400534391, Training Accuracy : 0.99976, Testing Accuracy : 0.9071

 86%|████████ | 259/300 [3:38:36<35:16, 51.62s/it]

Training Loss : 0.001034702947999467, Testing Loss : 0.7844615863984451, Training Accuracy : 0.9996, Testing Accuracy : 0.9095

87%|████████▋ | 260/300 [3:39:27<34:17, 51.43s/it]

Training Loss : 0.0014537495620222762, Testing Loss : 0.7962532920300961, Training Accuracy : 0.99964, Testing Accuracy : 0.9089

87%|████████▋ | 261/300 [3:40:17<33:09, 51.00s/it]

Training Loss : 0.0013783681384980447, Testing Loss : 0.7876159466482698, Training Accuracy : 0.99956, Testing Accuracy : 0.9096

87%|████████▋ | 262/300 [3:41:08<32:12, 50.85s/it]

Training Loss : 0.0012738354072213405, Testing Loss : 0.7890289969205856, Training Accuracy : 0.99958, Testing Accuracy : 0.9078

88%|████████▊ | 263/300 [3:41:58<31:18, 50.77s/it]

Training Loss : 0.0012937650906021008, Testing Loss : 0.7895704885629937, Training Accuracy : 0.99938, Testing Accuracy : 0.9094

88%|████████▊ | 264/300 [3:42:49<30:22, 50.64s/it]

Training Loss : 0.0013945443569641793, Testing Loss : 0.7974983543254435, Training Accuracy : 0.99954, Testing Accuracy : 0.9103

88%|████████▊ | 265/300 [3:43:39<29:25, 50.44s/it]

Training Loss : 0.0007057058783608955, Testing Loss : 0.7859823750948534, Training Accuracy : 0.99984, Testing Accuracy : 0.91

89%|████████▊ | 266/300 [3:44:29<28:31, 50.34s/it]

Training Loss : 0.001015762667848321, Testing Loss : 0.7953340827822685, Training Accuracy : 0.99978, Testing Accuracy : 0.9108

89%|████████▉ | 267/300 [3:45:19<27:42, 50.36s/it]

Training Loss : 0.0011850805660919286, Testing Loss : 0.8182892576366663, Training Accuracy : 0.99964, Testing Accuracy : 0.907

89%|████████▉ | 268/300 [3:46:10<26:54, 50.44s/it]

Training Loss : 0.0011611957864169381, Testing Loss : 0.8025971207916737, Training Accuracy : 0.99958, Testing Accuracy : 0.9106

 90%|█████████ | 269/300 [3:47:00<26:05, 50.50s/it]

Training Loss : 0.0011509560938633512, Testing Loss : 0.8312902440837352, Training Accuracy : 0.99964, Testing Accuracy : 0.9112

 90%|█████████ | 270/300 [3:47:51<25:18, 50.62s/it]

Training Loss : 0.0011922943215671694, Testing Loss : 0.8110762801946141, Training Accuracy : 0.9996, Testing Accuracy : 0.9106

 90%|█████████ | 271/300 [3:48:42<24:28, 50.65s/it]

Training Loss : 0.0011700592173519543, Testing Loss : 0.8089710786744952, Training Accuracy : 0.99952, Testing Accuracy : 0.9117

 91%|█████████ | 272/300 [3:49:33<23:37, 50.62s/it]

Training Loss : 0.0012257841811084654, Testing Loss : 0.7946078568324446, Training Accuracy : 0.99958, Testing Accuracy : 0.9121

 91%|█████████ | 273/300 [3:50:23<22:47, 50.65s/it]

Training Loss : 0.0008083832523954334, Testing Loss : 0.7955825499057769, Training Accuracy : 0.99982, Testing Accuracy : 0.9116

 91%|█████████▏| 274/300 [3:51:14<21:58, 50.71s/it]

Training Loss : 0.0009015704204631038, Testing Loss : 0.8067946028709412, Training Accuracy : 0.99978, Testing Accuracy : 0.913

 92%|█████████▏| 275/300 [3:52:05<21:07, 50.68s/it]

Training Loss : 0.0006721813988563372, Testing Loss : 0.8100527758598328, Training Accuracy : 0.99978, Testing Accuracy : 0.9133

 92%|█████████▏| 276/300 [3:52:55<20:10, 50.44s/it]

Training Loss : 0.0008692412797181168, Testing Loss : 0.8063401037454605, Training Accuracy : 0.9997, Testing Accuracy : 0.9115

 92%|█████████▏| 277/300 [3:53:44<19:15, 50.23s/it]

Training Loss : 0.0008091053482648567, Testing Loss : 0.8049347605824471, Training Accuracy : 0.9997, Testing Accuracy : 0.9145

93%|█████████| 278/300 [3:54:35<18:25, 50.23s/it]

Training Loss : 0.000782790182143217, Testing Loss : 0.8162846820831299, Training Accuracy : 0.99978, Testing Accuracy : 0.9129

93%|█████████| 279/300 [3:55:25<17:33, 50.18s/it]

Training Loss : 0.0006122961541143013, Testing Loss : 0.8013297542095185, Training Accuracy : 0.9998, Testing Accuracy : 0.9138

93%|█████████| 280/300 [3:56:15<16:42, 50.13s/it]

Training Loss : 0.0006124635356842191, Testing Loss : 0.8078599019568413, Training Accuracy : 0.99988, Testing Accuracy : 0.9134

94%|█████████| 281/300 [3:57:05<15:51, 50.07s/it]

Training Loss : 0.00047961143059961614, Testing Loss : 0.8196339421749115, Training Accuracy : 0.99988, Testing Accuracy : 0.9125

94%|█████████| 282/300 [3:57:54<14:59, 49.96s/it]

Training Loss : 0.0005316249108401825, Testing Loss : 0.8333961469046771, Training Accuracy : 0.99984, Testing Accuracy : 0.9103

94%|█████████| 283/300 [3:58:45<14:11, 50.10s/it]

Training Loss : 0.0007581076365476474, Testing Loss : 0.8343627868041397, Training Accuracy : 0.99976, Testing Accuracy : 0.9125

95%|█████████| 284/300 [3:59:36<13:25, 50.32s/it]

Training Loss : 0.0006309304761715612, Testing Loss : 0.8095015579223633, Training Accuracy : 0.9998, Testing Accuracy : 0.913

95%|█████████| 285/300 [4:00:26<12:35, 50.37s/it]

Training Loss : 0.0008398858370824018, Testing Loss : 0.7956558648586273, Training Accuracy : 0.99982, Testing Accuracy : 0.9127

95%|█████████| 286/300 [4:01:17<11:46, 50.44s/it]

Training Loss : 0.0009611545232563004, Testing Loss : 0.8115390979528427, Training Accuracy : 0.99968, Testing Accuracy : 0.9124

 96%|██████████| 287/300 [4:02:07<10:56, 50.51s/it]

Training Loss : 0.0007984618407156086, Testing Loss : 0.8114164962649345, Training Accuracy : 0.99978, Testing Accuracy : 0.9138

 96%|██████████| 288/300 [4:02:58<10:06, 50.51s/it]

Training Loss : 0.0003594461986311944, Testing Loss : 0.8026020537018776, Training Accuracy : 0.99992, Testing Accuracy : 0.9135

 96%|██████████| 289/300 [4:03:48<09:15, 50.52s/it]

Training Loss : 0.0005283019926857378, Testing Loss : 0.8130520912915469, Training Accuracy : 0.99984, Testing Accuracy : 0.9133

 97%|██████████| 290/300 [4:04:39<08:24, 50.40s/it]

Training Loss : 0.0005184437754675673, Testing Loss : 0.8091183502405882, Training Accuracy : 0.99988, Testing Accuracy : 0.9129

 97%|██████████| 291/300 [4:05:28<07:31, 50.16s/it]

Training Loss : 0.0006878233515386819, Testing Loss : 0.815845765221119, Training Accuracy : 0.99974, Testing Accuracy : 0.9145

 97%|██████████| 292/300 [4:06:18<06:40, 50.09s/it]

Training Loss : 0.0005505097410898088, Testing Loss : 0.8061463927447796, Training Accuracy : 0.99982, Testing Accuracy : 0.9147

 98%|██████████| 293/300 [4:07:08<05:50, 50.08s/it]

Training Loss : 0.000577539607511062, Testing Loss : 0.8225913215756416, Training Accuracy : 0.99984, Testing Accuracy : 0.9138

 98%|██████████| 294/300 [4:07:58<05:00, 50.09s/it]

Training Loss : 0.0006447613436704705, Testing Loss : 0.803214879655838, Training Accuracy : 0.99976, Testing Accuracy : 0.9137

 98%|██████████| 295/300 [4:08:48<04:10, 50.02s/it]

```
Training Loss : 0.0007302580217964714, Testing Loss :
0.8315870573401452, Training Accuracy : 0.99986, Testing Accuracy :
0.9131
```

```
 99%|██████████| 296/300 [4:09:38<03:19, 49.96s/it]
```

```
Training Loss : 0.0005535173174060765, Testing Loss :
0.8339561965703964, Training Accuracy : 0.99988, Testing Accuracy :
0.9115
```

```
 99%|██████████| 297/300 [4:10:28<02:30, 50.03s/it]
```

```
Training Loss : 0.0008971744713751832, Testing Loss :
0.8305504644989967, Training Accuracy : 0.99976, Testing Accuracy :
0.9122
```

```
 99%|██████████| 298/300 [4:11:19<01:40, 50.18s/it]
```

```
Training Loss : 0.00045974665993097003, Testing Loss :
0.8331555531620979, Training Accuracy : 0.99986, Testing Accuracy :
0.9116
```

```
100%|██████████| 299/300 [4:12:09<00:50, 50.30s/it]
```

```
Training Loss : 0.0007625088465726003, Testing Loss :
0.8300924083888531, Training Accuracy : 0.99972, Testing Accuracy :
0.9129
```

```
100%|██████████| 300/300 [4:13:00<00:00, 50.60s/it]
```

```
Training Loss : 0.0005087458202104608, Testing Loss :
0.8380652027606964, Training Accuracy : 0.99988, Testing Accuracy :
0.9138
```

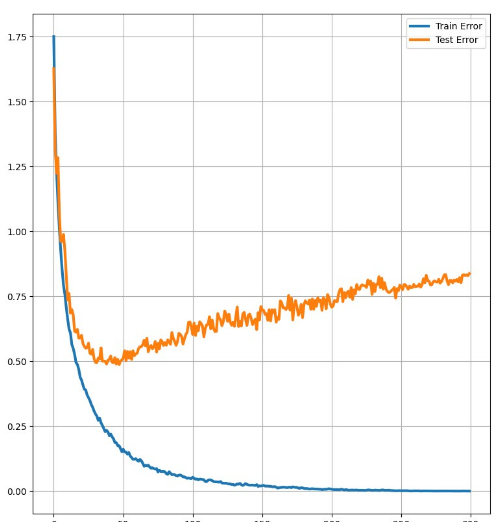
```python
print("Max Testing Accuracy: %s"%(max(test_accuracy)))
xmax = np.argmax(test_accuracy)
ymax = max(test_accuracy)
```

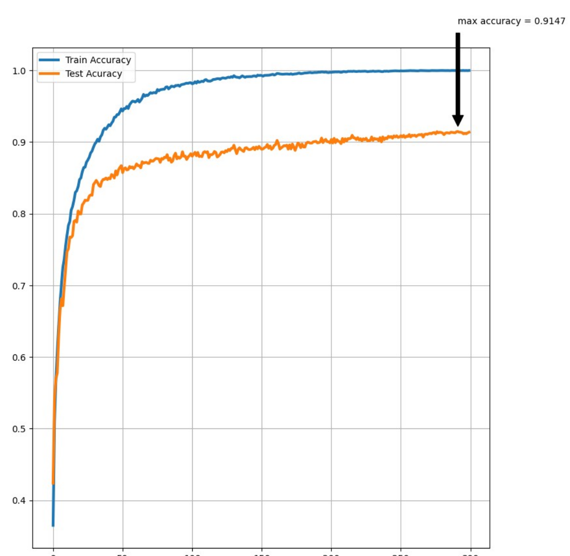
```
Max Testing Accuracy: 0.9147
```

Plotting the graph for train loss vs test loss and also for train accuracy vs test accuracy.

```python
# Create a figure object with two subplots, with a size of 20 by 10.
f, (fig1, fig2) = plt.subplots(1, 2, figsize=(20, 10))

# Set the number of data points in the training data to n.
n = len(train_loss)

# Plot the training loss and testing loss against the number of epochs
on the first subplot.
fig1.plot(range(n), train_loss, '-', linewidth='3', label='Train
```

```
Error')
fig1.plot(range(n), test_loss, '-', linewidth='3', label='Test Error')

# Plot the training accuracy and testing accuracy against the number
of epochs on the second subplot.
fig2.plot(range(n), train_accuracy, '-', linewidth='3', label='Train
Accuracy')
fig2.plot(range(n), test_accuracy, '-', linewidth='3', label='Test
Acuracy')

# Annotate the maximum accuracy achieved with an arrow on the second
subplot.
fig2.annotate('max accuracy = %s'%(ymax), xy=(xmax, ymax),
xytext=(xmax, ymax+0.15), arrowprops=dict(facecolor='black',
shrink=0.05))

# Turn on the grid lines for both subplots.
fig1.grid(True)
fig2.grid(True)

# Add legends to both subplots.
fig1.legend()
fig2.legend()

# Save the figure to a file named "trainTestCurve.png".
f.savefig("./trainTestCurve.png")
```

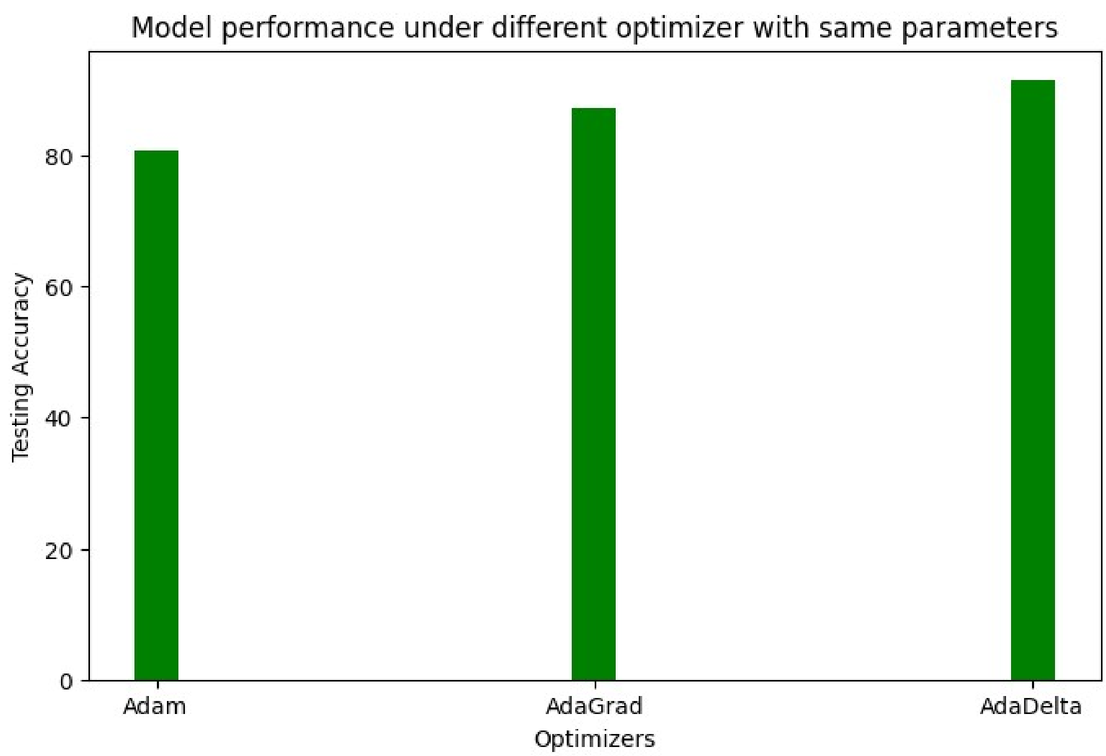





```
torch.save(model.state_dict(), '/content/model1.pt')
```

We tried our model with 3 different optimizer, Adam, AdaGrad and AdaDelta, and below is the bar plot with their accuracy.

```python
import matplotlib.pyplot as plt

# creating the dataset
data = {'Adam':80.67, 'AdaGrad':87.23, 'AdaDelta':91.47}
optimizer = list(data.keys())
accuracy = list(data.values())

fig = plt.figure(figsize = (8, 5))

# creating the bar plot
plt.bar(optimizer, accuracy, color ='green',
        width = 0.1)

plt.xlabel("Optimizers")
plt.ylabel("Testing Accuracy")
plt.title("Model performance under different optimizer with same
parameters")
plt.show()
```



Below we are just converting our pt model into onnx format to build the Resnet Architecture diagram

```
pip install onnx
```

```
Looking in indexes: https://pypi.org/simple, https://us-
python.pkg.dev/colab-wheels/public/simple/
Collecting onnx
```

```
  Downloading onnx-1.13.1-cp39-cp39-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (13.5 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.5/13.5 MB 91.5 MB/s eta
0:00:00
ent already satisfied: typing-extensions>=3.6.2.1 in
/usr/local/lib/python3.9/dist-packages (from onnx) (4.5.0)
Requirement already satisfied: protobuf<4,>=3.20.2 in
/usr/local/lib/python3.9/dist-packages (from onnx) (3.20.3)
Requirement already satisfied: numpy>=1.16.6 in
/usr/local/lib/python3.9/dist-packages (from onnx) (1.22.4)
Installing collected packages: onnx
Successfully installed onnx-1.13.1
```

```
model = resnet_model()
model.load_state_dict(torch.load('/content/model1.pt'))
# set the model to inference mode
model.eval()

# Let's create a dummy input tensor
dummy_input = torch.randn(4, 3, 32, 32)
# torch.onnx.export(model, dummy_input, "final_model.onnx")

# Export the model
torch.onnx.export(model,         # model being run
      dummy_input,       # model input (or a tuple for multiple
inputs)
      "final_model1.onnx",       # where to save the model
      export_params=True,  # store the trained parameter weights
inside the model file
      opset_version=13,    # the ONNX version to export the model to
      do_constant_folding=True,  # whether to execute constant folding
for optimization
      input_names = ['modelInput'],   # the model's input names
      output_names = ['modelOutput'],# the model's output names
)

torch.save(model.state_dict(), 'final_model.pt')
```

```
Model Config:
-------------------------------------
N (# Layers)    : 4
B (# Blocks)    : [3, 3, 2, 3]
C (# Channels)  : [64, 128, 128, 256]
F (Conv Kernel) : [3, 3, 3, 3]
K (Skip Kernel) : [1, 1, 1, 1]
P (Pool Kernel) : 4
============= Diagnostic Run torch.onnx.export version 2.0.0+cu118
=============
```

```
verbose: False, log level: Level.ERROR
======================= 0 NONE 0 NOTE 0 WARNING 0 ERROR
========================
```

```python
from google.colab import files
files.download('final_model1.onnx')
```

```
<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```

```python
import urllib.request
from IPython.display import SVG, display

# URL of the SVG image we are downloading
url = "https://svgshare.com/i/s0w.svg"

# Download the SVG image
svg_data = urllib.request.urlopen(url).read()

# Display the SVG image in Colab
display(SVG(svg_data))
```