In this document, The steps of the program execution:

STEP 1: Importing the essential libraries to run the program

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
import torch
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
```

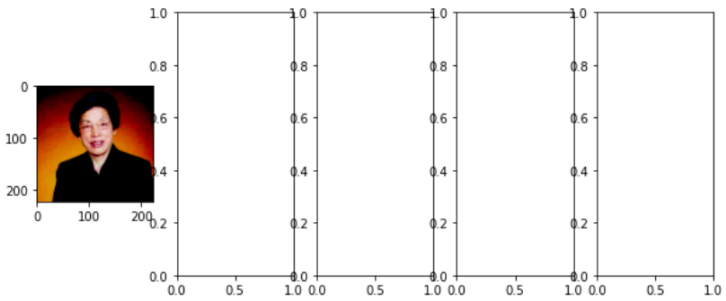
STEP 2:
Loading the test and train data in the notebook. The test_train_data() function loads the data in the tensor format. train_imshow() will print a few of the trained images
Note: this is just to check the training images and has no other purpose

```
#Importing all the dataset from directory to the notebook
def test_train_data():
    # data_dir = 'C:\\Users\\16124\\Documents\\Rishika\\Internship work\\trainset'

    transform = transforms.Compose([transforms.Resize(256),transforms.CenterCrop(224),transforms.ToTensor(),transforms.Normalize((0.5, 0.5, 0.5),
                                                                                                                                  (0.5, 0.5, 0.5))])

    train_set = datasets.ImageFolder( '/content/drive/MyDrive/Program/train_01', transform =transform )
    test_set = datasets.ImageFolder( '/content/drive/MyDrive/Program/test_01', transform =transform)

    train = DataLoader(train_set, batch_size=1, shuffle=True)
    test = DataLoader(test_set, batch_size=1, shuffle=True)
    return train,test

#Printing few images to check the training set
def train_imshow():
    dataiter = iter(train)
    images, labels = dataiter.next()
    fig, axes = plt.subplots(figsize=(10, 4), ncols=5)
    for i in range(5):
        ax = axes[i]
        ax.imshow(images[i].permute(1, 2, 0))
    plt.imshow()
```

STEP 3: Printing a few training set images

```
#Assigning variable names for the test and training data sets
train,test = test_train_data()
```

```
train_imshow()
```



STEP 4:
Building a CNN network that will be used to train the model.

```
class Unit(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(Unit, self).__init__()

        self.conv = nn.Conv2d(in_channels=in_channels, kernel_size=3, out_channels=out_channels, stride=1, padding=1)
        self.bn = nn.BatchNorm2d(num_features=out_channels)
        self.relu = nn.ReLU()

    def forward(self, input):
        output = self.conv(input)
        output = self.bn(output)
        output = self.relu(output)

        return output


class SimpleNet(nn.Module):
    def __init__(self,num_classes=1012):
        super(SimpleNet,self).__init__()

        #Creating 14 layers of the unit with max pooling in between
        self.unit1 = Unit(in_channels=3,out_channels=32)
        self.unit2 = Unit(in_channels=32, out_channels=32)
        self.unit3 = Unit(in_channels=32, out_channels=32)
```

STEP 5:
Assigning variable names to the network to make it callable and setting the appropriate optimizer and loss function to train the model

```
model = SimpleNet()
```

```
net = SimpleNet()
```

```
# Checking to see if gpu support is available
cuda_avail = torch.cuda.is_available()
print("true")
# Creating model, optimizer and loss function
model = SimpleNet(num_classes=1012)

#if cuda is available, moving the model to the GPU (which is not happening in my system)
if cuda_avail:
    model.cuda()
    net.cuda()
    print("done")
#Define the optimizer and loss function
optimizer = optim.Adam(model.parameters(), lr=0.001, weight_decay=0.0001)
loss_fn = nn.CrossEntropyLoss()
```

```
true
done
```

STEP 6:
A function called adjust_learning_rate() will adjust the learning rate based on the epoch given

```
# Creating a learning rate adjustment function that divides the learning rate by 10 every 30 epochs
def adjust_learning_rate(epoch):
    lr = 0.001

    if epoch > 180:
        lr = lr / 1000000
    elif epoch > 150:
        lr = lr / 100000
    elif epoch > 120:
        lr = lr / 10000
    elif epoch > 90:
        lr = lr / 1000
    elif epoch > 60:
        lr = lr / 100
    elif epoch > 30:
        lr = lr / 10

    for param_group in optimizer.param_groups:
        param_group["lr"] = lr
```

STEP 7: When called, the save_models function will save our trained model and the test_model() will test the trained model on the test samples.

```
[46] # A  callable function to save the trained model
     def save_models(epoch):
         PATH = '/content/drive/MyDrive/Program/new_model.pth'
         torch.save(model.state_dict(),PATH.format(epoch))
         print("Checkpoint saved")
```

```
# A callable function which will evaluate trained model on the test data
def test_model():
    model.eval()
    test_acc = 0.0
    for i, (images, labels) in enumerate(test):

        if cuda_avail:
            images = Variable(images.cuda())
            labels = Variable(labels.cuda())

        # Predicting classes using images from the test set
        outputs = model(images)
        _, prediction = torch.max(outputs.data, 1)
        test_acc += torch.sum(prediction == labels.data)

    # Computing the average acc and loss over all the test images
    test_acc = test_acc / len(test)

    return test_acc
```

STEP 8:
Train_model will train and model along with saving and testing it on the test set by calling the respective functions.

```
def train_model(num_epochs):
    best_acc = 0.0
    for epoch in range(num_epochs):
        model.train()
        train_acc = 0.0
        train_loss = 0.0
        for i, (images, labels) in enumerate(train):
          # Moving images and labels to gpu if available
          images = Variable(images.cuda())
          labels = Variable(labels.cuda())

          # Clearing all accumulated gradients
          optimizer.zero_grad()

          # Predicting classes using images from the test set
          outputs = model(images)

          # Computing the loss based on the predictions and actual labels
          loss = loss_fn(outputs, labels)

          # Backpropagating the loss
          loss.backward()

          # Adjusting parameters according to the calculated gradients
          optimizer.step()
          train_loss += loss.item()* images.size(0)
          _, prediction = torch.max(outputs.data, 1)
          train_acc += torch.sum(prediction == labels.data)

        # Calling the learning rate adjustment function
        adjust_learning_rate(epoch)
```

STEP 9:
This step will call the train model function and pass the number of epochs to be done

```
train_model(200)
```

```
Epoch 0, Train Accuracy: 0.0 , TrainLoss: 7.331789747014776 , Test Accuracy: 0.0
Epoch 1, Train Accuracy: 0.004095003940165043 , TrainLoss: 6.9496637269378585 , Test Accuracy: 0.0
Checkpoint saved
Epoch 2, Train Accuracy: 0.019656019285321236 , TrainLoss: 5.9388588273456895 , Test Accuracy: 0.01785714365541935
Checkpoint saved
Epoch 3, Train Accuracy: 0.23177723586559296 , TrainLoss: 3.536463046547126 , Test Accuracy: 0.0714285746216774
Epoch 4, Train Accuracy: 0.5880426168441772 , TrainLoss: 1.6712801212268371 , Test Accuracy: 0.0357142873108387
Checkpoint saved
Epoch 5, Train Accuracy: 0.8271908164024353 , TrainLoss: 0.7150875018199688 , Test Accuracy: 0.1071428656578064
```

STEP 10:
This is the part where we evaluate our model on the hidden test set. The steps done are shown below.

Inference with the Saved Model

```
# Steps done here are as follows:
# 1. Loading our saved model
# 2. Creating a function called predict_image which will read the image in the PIL format and convert it into
#    tensor format so that further comparasion can be done
# 3. Another function called train_pred is used to compare the returned index from test data witht hte trained data
#    and give an output matched or not matched
# 4. In the last step we predict the confidence score of the prediction
# Us as a user just have to provide the path for the test image
```

```
[188] #Loading the saved model
      path = '/content/drive/MyDrive/Program/new_model.pth'
      checkpoint = torch.load(path)
      model = SimpleNet(num_classes=1012)


      model.load_state_dict(checkpoint)
      model.eval()
```

STEP 11:

```
# Reads and converts the images from PIL format image to tensor format and returns an index value which will be
# compared with train data later
from PIL import Image
from torch.autograd import Variable
import torchvision
def predict_image(image_path):
    print("Prediction in progress")
    image = Image.open(image_path)

    # Define transformations for the image, should (note that imagenet models are trained with image size 224)
    transformation = transforms.Compose([transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

    # Preprocess the image
    image_tensor = transformation(image).float()

    # Add an extra batch dimension since pytorch treats all images as batches
    image_tensor = image_tensor.unsqueeze_(0)


    # Turn the input into a Variable
    input = Variable(image_tensor)
    input = input.to(device)
#     print('input',input)
    # Predict the class of the image
    input.cuda()
    output = net(input)
    index = output.data.cpu().numpy().argmax()


    # Displaying the test image
    plt.imshow(image)

    return index
```

STEP 12:

This function compares the index value of the test set with the trained data and gives an output matched if the label and index match.

```
# This callable function takes the test data index and compares it with the trained set and compares the
# information and displays a match or no match
def train_pred(index):
    count = 0
    for data in train:
        images,labels = data
        images, labels = images.cuda(), labels.cuda()
        if labels == index:
            print("output matched",labels)

        else:
            count += 1
```

STEP 13:
Here, the user has to put the Test Image directory in the image_path variable and then run the cell

```
# User can put the directory of the image to be tested in the variable image_path and run the cell
image_path = "/content/drive/MyDrive/Program/train_01/0005_0000530/0005_0000530_script.jpg"
index = predict_image(image_path)
```

STEP 14:

```
# Calling the function for comparison
train_pred(index)
```

```
output matched tensor([857])
```

STEP 15:
Accuracy of the model:

```
#Accuracy prediction
r = 0
s = 0
with torch.no_grad():
    for data in test:
        images,test_label = data
        r += test_label.size(0)
        for info in train:
            images,train_label =info
            if train_label == test_label:
                s += (train_label == test_label).sum().item()

print('Accuracy of a network is: %d %% ' % (100*r/s))
```

```
Accuracy of a network is: 73 %
```