Colab Link

Colab Link:https://colab.research.google.com/drive/1MGrC6072x2grj2Xa40JtaYlnTeD8tik2?usp=sharing

In [ ]:
```
# Importing Everyhing We Need
from google.colab import drive
from skimage.transform import resize
from sklearn.model_selection import train_test_split
from torch.utils.data import DataLoader
from torch.utils.data.sampler import SubsetRandomSampler
from torchvision import datasets, models, transforms
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
import shutil
import time
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import random
import PIL
```

In [ ]:
```
#mount the drive
from google.colab import drive
drive.mount("/content/gdrive", force_remount=True)
```

```
Mounted at /content/gdrive
```

- From this point, we have gathered data from the stanford dog breed data set of more than 12000 images containing 120 dog species. All data is stored in google drive.
- For the purpuse of testing, we have choosen a small portion of the data, containg only 992 samples of 5 species to reduce training time.

In [ ]:
```
# Define Necessary Settings
small_test_path = '/content/gdrive/MyDrive/aps360smalltest'
img_width, img_height = 224, 224
channels = 3
batch_size = 64
num_images= 992
image_arr_size= img_width * img_height * channels
```

In [ ]:
```
cd /content/gdrive/MyDrive/aps360smalltest
```

```
/content/gdrive/.shortcut-targets-by-id/1H87uh8DLPkPkphCHYbyvF93o1YLh2xPo/aps360smalltest
```

In [ ]:
```
# Define the paths
small_test_folder = '.'  # Current directory
small_test_train_folder = 'train'
```

```
small_test_val_folder = 'val'
small_test_test_folder = 'test'
```

In [ ]:
```
first_time = False
```

In [ ]:
```
# Iterate through each class folder to split the images into train test and val
# WillOnly execute for the 1st time
# Any following times running will show "images already split"

if first_time:

  print("Splitting Images into 70% training, 15% validation, 15% testing")
  for class_folder in os.listdir(small_test_folder):
      class_path = os.path.join(small_test_folder, class_folder)
      if os.path.isdir(class_path):  # Check if it's a directory (class folder)

          # Create corresponding class folders in train, val, test
          os.makedirs(os.path.join(small_test_train_folder, class_folder), exist_ok
          os.makedirs(os.path.join(small_test_val_folder, class_folder), exist_ok=T
          os.makedirs(os.path.join(small_test_test_folder, class_folder), exist_ok=

          # Get the list of image files for this class
          image_files = [f for f in os.listdir(class_path) if os.path.isfile(os.pat

          # Split into train and temp (test + val)
          train_files, temp_files = train_test_split(image_files, test_size=0.3, ra

          # Split temp into val and test
          val_files, test_files = train_test_split(temp_files, test_size=0.5, randc

          # Copy the files to their respective folders
          for file in train_files:
              shutil.copy(os.path.join(class_path, file), os.path.join(small_test_t
          for file in val_files:
              shutil.copy(os.path.join(class_path, file), os.path.join(small_test_v
          for file in test_files:
              shutil.copy(os.path.join(class_path, file), os.path.join(small_test_t

  first_time = False

else:
  print("Images Already Split")
```

```
Images Already Split
```

In [ ]:
```
# Define the paths again now that they are created
train_folder = '/content/gdrive/MyDrive/aps360smalltest/train'
val_folder = '/content/gdrive/MyDrive/aps360smalltest/val'
test_folder = '/content/gdrive/MyDrive/aps360smalltest/test'

# Count the images to check if we split it correctly
def count_images(folder):
    count = 0
    for class_folder in os.listdir(folder):
        class_path = os.path.join(folder, class_folder)
        if os.path.isdir(class_path):
            count += len([f for f in os.listdir(class_path) if os.path.isfile(os.pa
    return count

train_count = count_images(train_folder)
val_count = count_images(val_folder)
test_count = count_images(test_folder)
```

```
total_count = train_count + val_count + test_count

# Print the number of images for train, test and val
print("Number of training images:", train_count, "({:.2f}%)".format(train_count/tot
print("Number of validation images:", val_count, "({:.2f}%)".format(val_count/total
print("Number of testing images:", test_count, "({:.2f}%)".format(test_count/total_
```

```
Number of training images: 699 (69.83%)
Number of validation images: 150 (14.99%)
Number of testing images: 152 (15.18%)
```

- Now we have split the images into 70% training, 15% validation and 15% testing data
- We create the data loader for training. Data Transform ensures all images are croped to same size

In [ ]:
```
# data transformations to crop the images into same size of 224*224
data_transform = transforms.Compose([
    transforms.Resize((224, 224)),  # Resize to match CNN input
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])  # Normalize us
])


#change imagefolder class_to_idx and samples to avoid mismatching of labels to imag
trainset.classes = breeds
valset.classes = breeds
testset.classes = breeds

trainset.class_to_idx = breeds_to_idx
valset.class_to_idx = breeds_to_idx
testset.class_to_idx = breeds_to_idx

for sample in trainset.samples:
  breed_id = 0
  while breed_id < num_classes:
    if breeds[breed_id] in sample[0]:
      right_tuple = (sample[0], breed_id)
      correct_train_sample.append(right_tuple)
    breed_id+=1

trainset.samples = correct_train_sample

for sample in testset.samples:
  breed_id = 0
  while breed_id < num_classes:
    if breeds[breed_id] in sample[0]:
      right_tuple = (sample[0], breed_id)
      correct_test_sample.append(right_tuple)
    breed_id+=1

testset.samples = correct_test_sample

for sample in valset.samples:
  breed_id = 0
  while breed_id < num_classes:
    if breeds[breed_id] in sample[0]:
      right_tuple = (sample[0], breed_id)
      correct_val_sample.append(right_tuple)
    breed_id+=1

valset.samples = correct_val_sample
```

```
# Load training data
train_dataset = datasets.ImageFolder(root='train', transform=data_transform)
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)

# Load validation data
val_dataset = datasets.ImageFolder(root='val', transform=data_transform)
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)

# Load test data
test_dataset = datasets.ImageFolder(root='test', transform=data_transform)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
```

- Now that we have the data loaders, we want to write and test a simple CNN baseline model for our future reference.

In [ ]:
```
# In this case of small data set, we only have 5 class of dogs
num_classes = 5

# This is the simple cnn we use for baseline model

class SimpleCNN(nn.Module):
  def __init__(self):
    super(SimpleCNN, self).__init__()
    self.name = "Simple_Net"
    self.conv1 = nn.Conv2d(3, 16, kernel_size=3, padding=1)
    self.conv2 = nn.Conv2d(16, 32, kernel_size=3, padding=1)
    self.pool = nn.MaxPool2d(2, 2)
    self.fc1 = nn.Linear(32 * 56 * 56, 128) # 56 comes from [(224-3+2)/2 -3+2]/2, 3
    self.fc2 = nn.Linear(128, num_classes)

  def forward(self, x):
    x = self.pool(F.relu(self.conv1(x)))
    x = self.pool(F.relu(self.conv2(x)))
    x = x.view(-1, 32 * 56 * 56)
    x = F.relu(self.fc1(x))
    x = self.fc2(x)
    return x
```

In [ ]:
```
# Training
# Those are from lab4
def get_model_name(name, batch_size, learning_rate, epoch):
    """ Generate a name for the model consisting of all the hyperparameter values

    Args:
        config: Configuration object containing the hyperparameters
    Returns:
        path: A string with the hyperparameter name and value concatenated
    """
    path = "model_{0}_bs{1}_lr{2}_epoch{3}".format(name,
                                                   batch_size,
                                                   learning_rate,
                                                   epoch)
    return path
```

In [ ]:
```
def normalize_label(labels):
    """
    Given a tensor containing 2 possible values, normalize this to value between 0/

    Args:
        labels: a 1D tensor containing two possible scalar values
```

```
    Returns:
        A tensor normalize to value between 0/1
    """
    max_val = torch.max(labels)
    min_val = torch.min(labels)
    norm_labels = (labels - min_val)/(max_val - min_val)
    return norm_labels
```

In [ ]:
```
def evaluate(net, loader, criterion):
    """ Evaluate the network on the validation set.

     Args:
         net: PyTorch neural network object
         loader: PyTorch data loader for the validation set
         criterion: The loss function
     Returns:
         err: A scalar for the avg classification error over the validation set
         loss: A scalar for the average loss function over the validation set
     """
    total_loss = 0.0
    total_err = 0.0
    total_epoch = 0
    for i, data in enumerate(loader, 0):
        # Get the inputs
        inputs, labels = data
        # Send data to device
        inputs, labels = inputs.to(device), labels.to(device)

        outputs = net(inputs)
        loss = criterion(outputs, labels.long())

        # Get predicted class indices
        _, predicted = torch.max(outputs.data, 1)
        # Compare predicted with actual labels
        corr = predicted != labels
        total_err += int(corr.sum())
        total_loss += loss.item()
        total_epoch += len(labels)
    err = float(total_err) / total_epoch
    loss = float(total_loss) / (i + 1)
    return err, loss
```

In [ ]:
```
# Training Curve
def plot_training_curve(path):
    """ Plots the training curve for a model run, given the csv files
    containing the train/validation error/loss.

    Args:
        path: The base path of the csv files produced during training
    """
    import matplotlib.pyplot as plt
    train_err = np.loadtxt("{}_train_err.csv".format(path))
    val_err = np.loadtxt("{}_val_err.csv".format(path))
    train_loss = np.loadtxt("{}_train_loss.csv".format(path))
    val_loss = np.loadtxt("{}_val_loss.csv".format(path))
    plt.title("Train vs Validation Error")
    n = len(train_err) # number of epochs
    plt.plot(range(1,n+1), train_err, label="Train")
    plt.plot(range(1,n+1), val_err, label="Validation")
    plt.xlabel("Epoch")
    plt.ylabel("Error")
    plt.legend(loc='best')
    plt.show()
```

```
    plt.title("Train vs Validation Loss")
    plt.plot(range(1,n+1), train_loss, label="Train")
    plt.plot(range(1,n+1), val_loss, label="Validation")
    plt.xlabel("Epoch")
    plt.ylabel("Loss")
    plt.legend(loc='best')
    plt.show()
```

In [ ]:

```
# Train net function, changed from lab 4
def train_net(net, batch_size=64, learning_rate=0.01, num_epochs=30):

    # Define data transformations
    data_transform = transforms.Compose([
        transforms.Resize((224, 224)),  # Resize to match CNN input
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])  # Normaliz
    ])

    # Load training data
    train_dataset = datasets.ImageFolder(root='train', transform=data_transform)
    train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)

    # Load validation data
    val_dataset = datasets.ImageFolder(root='val', transform=data_transform)
    val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)

    # Load test data
    test_dataset = datasets.ImageFolder(root='test', transform=data_transform)
    test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

    # Initialize the model
    model = SimpleCNN()  # Replace SimpleCNN with your actual model class
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model.to(device)

    # Define loss function and optimizer
    criterion = nn.CrossEntropyLoss()
    optimizer =  optim.SGD(net.parameters(), lr=learning_rate, momentum=0.9)# Adjus


    # Set up numpy arrays to store training/validation loss/accuracy
    train_err = np.zeros(num_epochs)
    train_loss = np.zeros(num_epochs)
    val_err = np.zeros(num_epochs)
    val_loss = np.zeros(num_epochs)

    start_time = time.time()
    for epoch in range(num_epochs):  # loop over the dataset multiple times
        total_train_loss = 0.0
        total_train_err = 0.0
        total_epoch = 0
        for i, data in enumerate(train_loader, 0):
            # Get the inputs
            inputs, labels = data
            labels = normalize_label(labels) # Convert labels to 0/1
            # Zero the parameter gradients
            optimizer.zero_grad()
            # Forward pass, backward pass, and optimize
            outputs = net(inputs)
            loss = criterion(outputs, labels.long())
            loss.backward()
            optimizer.step()
             # Calculate the statistics
            _, predicted = torch.max(outputs.data, 1) # Get predicted class index
```

```
            corr = predicted != labels.long() # Compare predicted class with actual
            total_train_err += int(corr.sum())
            total_train_loss += loss.item()
            total_epoch += len(labels)
        train_err[epoch] = float(total_train_err) / total_epoch
        train_loss[epoch] = float(total_train_loss) / (i+1)
        val_err[epoch], val_loss[epoch] = evaluate(net, val_loader, criterion)
        print(("Epoch {}: Train err: {}, Train loss: {} |"+
               "Validation err: {}, Validation loss: {}").format(
                   epoch + 1,
                   train_err[epoch],
                   train_loss[epoch],
                   val_err[epoch],
                   val_loss[epoch]))
        # Save the current model (checkpoint) to a file
        model_path = get_model_name(net.name, batch_size, learning_rate, epoch)
        torch.save(net.state_dict(), model_path)
    print('Finished Training')
    end_time = time.time()
    elapsed_time = end_time - start_time
    print("Total time elapsed: {:.2f} seconds".format(elapsed_time))

    # Write the train/test loss/err into CSV file for plotting later
    epochs = np.arange(1, num_epochs + 1)
    np.savetxt("{}_train_err.csv".format(model_path), train_err)
    np.savetxt("{}_train_loss.csv".format(model_path), train_loss)
    np.savetxt("{}_val_err.csv".format(model_path), val_err)
    np.savetxt("{}_val_loss.csv".format(model_path), val_loss)
```

In [ ]:
```
# Initialize the model
model1 = SimpleCNN()  # Replace SimpleCNN with your actual model class
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model1.to(device)
```

Out[ ]:
```
SimpleCNN(
  (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=Fals
e)
  (fc1): Linear(in_features=100352, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=5, bias=True)
)
```

In [ ]:
```
use_cuda = True
torch.cuda.empty_cache()
```

In [ ]:
```
# The function get accuracy, we will be using it to get the accuracy of our simple
def get_accuracy(model, data_loader):
    correct = 0
    total = 0
    with torch.no_grad(): # disable gradient calculation
        for imgs, labels in data_loader:
            if use_cuda and torch.cuda.is_available():
                imgs = imgs.cuda()
                labels = labels.cuda()
            output = model(imgs)
            pred = output.max(1, keepdim=True)[1]
            correct += pred.eq(labels.view_as(pred)).sum().item()
            total += imgs.shape[0]
    return correct / total
```

In [ ]:
```
get_accuracy(model1, val_loader)
```

Out[ ]:
```
0.2
```

- Now we see what accuracy our simple cnn is approximitely 20%, which is low.
- It is also reasonable since this is a very simple CNN with only 2 convolutional layers, 1 max pooling layer and 2 linear layers.
- This will act as our baseline model. The model we eventually build should be at least better than this.

- Now we proceed to building our own CNN from scratch

In [ ]:
```
class BreedClassifier(nn.Module):
    def __init__(self):
        super(BreedClassifier, self).__init__()

        self.conv1 = nn.Conv2d(3, 32, 3, padding=1)
        self.conv2 = nn.Conv2d(32, 64, 3, padding=1)
        self.conv3 = nn.Conv2d(64, 128, 3, padding=1)
        self.conv4 = nn.Conv2d(128, 256, 3, padding=1)
        self.conv5 = nn.Conv2d(256, 512, 3, padding=1)

        self.pool = nn.MaxPool2d(2, 2)

        self.fc1 = nn.Linear(512 * 56 *56, 1024)
        self.fc2 = nn.Linear(1024, 512)
        self.fc3 = nn.Linear(512, num_classes)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))
        x = self.pool(F.relu(self.conv4(x)))
        x = self.pool(F.relu(self.conv5(x)))

        x = x.view(-1, 512 * 56 * 56)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)

        return x
```

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

```
Mounted at /content/drive
```

In [ ]:

In [ ]:

- Now we see how much accuracy our CNN built from scrach can get, we aim to maximize the performance of our task. We introduce transfer learning. Among those pretrained CNNs in the database, we find Alexnet most fit the purpose of our training. So we would perform transfer learing on Alexnet

In [ ]:
```
# Define necessary settings
master_path = '/content/gdrive/MyDrive/aps360smalltest'
```

```
img_width, img_height = 224, 224
channels = 3
batch_size = 64
num_images= 992
image_arr_size= img_width * img_height * channels
```

In [ ]:
```
cd /content/gdrive/MyDrive/aps360smalltest
```

```
/content/gdrive/.shortcut-targets-by-id/1H87uh8DLPkPkphCHYbyvF93o1YLh2xPo/aps360sm
alltest
```

In [ ]:
```
# Define the paths
image_folder = '.'
train_folder = 'train'
val_folder = 'val'
test_folder = 'test'
```

In [ ]:
```
# Iterate through each class folder to split the images into train test and val
# Will Only execute for the 1st time
# Any following times running will show "images already split"

if first_time:

  print("Splitting Images into 70% training, 15% validation, 15% testing")
  for class_folder in os.listdir(small_test_folder):
      class_path = os.path.join(small_test_folder, class_folder)
      if os.path.isdir(class_path):  # Check if it's a directory (class folder)

          # Create corresponding class folders in train, val, test
          os.makedirs(os.path.join(small_test_train_folder, class_folder), exist_ok
          os.makedirs(os.path.join(small_test_val_folder, class_folder), exist_ok=T
          os.makedirs(os.path.join(small_test_test_folder, class_folder), exist_ok=

          # Get the list of image files for this class
          image_files = [f for f in os.listdir(class_path) if os.path.isfile(os.pat

          # Split into train and temp (test + val)
          train_files, temp_files = train_test_split(image_files, test_size=0.3, ra

          # Split temp into val and test
          val_files, test_files = train_test_split(temp_files, test_size=0.5, randc

          # Copy the files to their respective folders
          for file in train_files:
              shutil.copy(os.path.join(class_path, file), os.path.join(small_test_t
          for file in val_files:
              shutil.copy(os.path.join(class_path, file), os.path.join(small_test_v
          for file in test_files:
              shutil.copy(os.path.join(class_path, file), os.path.join(small_test_t

  first_time = False

else:
  print("Images Already Split")
```

```
Images Already Split
```

In [ ]:
```
# Define the paths again now that they have been created
train_folder = '/content/gdrive/MyDrive/aps360smalltest/train'
val_folder = '/content/gdrive/MyDrive/aps360smalltest/val'
test_folder = '/content/gdrive/MyDrive/aps360smalltest/test'

# Count the images to check if we split it correctly
def count_images(folder):
```

```
    count = 0
    for class_folder in os.listdir(folder):
        class_path = os.path.join(folder, class_folder)
        if os.path.isdir(class_path):
            count += len([f for f in os.listdir(class_path) if os.path.isfile(os.pa
    return count

train_count = count_images(train_folder)
val_count = count_images(val_folder)
test_count = count_images(test_folder)


total_count = train_count + val_count + test_count

# Print the number of images for train, test and val
print("Number of training images:", train_count, "({:.2f}%)".format(train_count/tot
print("Number of validation images:", val_count, "({:.2f}%)".format(val_count/total
print("Number of testing images:", test_count, "({:.2f}%)".format(test_count/total_
```

```
Number of training images: 699 (69.83%)
Number of validation images: 150 (14.99%)
Number of testing images: 152 (15.18%)
```

In [ ]:
```
# Define data transformations
data_transform = transforms.Compose([
    transforms.Resize((224, 224)),  # Resize to match CNN input
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])  # Normalize us
])

# Load training data
train_dataset = datasets.ImageFolder(root='train', transform=data_transform)
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)

# Load validation data
val_dataset = datasets.ImageFolder(root='val', transform=data_transform)
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)

# Load test data
test_dataset = datasets.ImageFolder(root='test', transform=data_transform)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
```

- Now we introduce Alexnet

In [ ]:
```
num_classes = 20
```

In [ ]:
```
alexnet = models.alexnet(pretrained=True)

#freeze model parameters, those pretrained alexnet parameters won't be updated duri
for param in alexnet.parameters():
  param.requires_grad = False

# We are adding final layers to AlexNet to fit our tasks for Transfer Learning
# In this case, since we are using small net data with only 5 classes,
# we are using small hidden layer size of 512 and drop out rate
# of 0.5 to make the model more accurate

# The structre of the modyfied alexnet is refernced to this following article:
# https://github.com/kipkoechjosh/APS360-Dog-Breed-Classifier/blob/main/final_dogs_
# After careful tunning of the neruals of each layer we decided to keep the followi

alexnet.classifier.add_module("4", nn.Linear(4096, 512))
```

```
alexnet.classifier.add_module("6", nn.Dropout(p=0.5, inplace=False))
alexnet.classifier.add_module("7", nn.Linear(512, 256))
alexnet.classifier.add_module("8", nn.ReLU(inplace=True))
alexnet.classifier.add_module("9", nn.Linear(256, num_classes))
alexnet
```

Out[ ]:

```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=Fals
e)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=Fals
e)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=Fals
e)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=512, bias=True)
    (5): ReLU(inplace=True)
    (6): Dropout(p=0.5, inplace=False)
    (7): Linear(in_features=512, out_features=256, bias=True)
    (8): ReLU(inplace=True)
    (9): Linear(in_features=256, out_features=20, bias=True)
  )
)
```

- Now we have made revelent modifications to Alexnet to fit our model, we implement the get accuracy function and the train function
- the get_accuracy_of_transfer and train helper functions are taken reference and credited to this following article: https://github.com/kipkoechjosh/APS360-Dog-Breed-Classifier/blob/main/final_dogs_breed_classifier_with_transfer_learning.ipynb

In [ ]:

```
def get_accuracy_of_transfer(model, train=False):
    if train:
        data_loader = train_loader
    else:
        data_loader = val_loader

    # Set the model to evaluation mode
    model.eval()
    correct = 0
    total = 0
    for imgs, labels in data_loader:

        #Enable GPU
        if use_cuda and torch.cuda.is_available():
```

```
            imgs = imgs.cuda()
            labels = labels.cuda()

        output = model(imgs)

        # Get predicted class index
        pred = output.max(1, keepdim=True)[1]
        correct += pred.eq(labels.view_as(pred)).sum().item()
        total += imgs.shape[0]
    return correct / total
```

In [ ]:

```
# Here is the train function for alexnet transfer learning. We will be training thi

def train(model, model_name, learning_rate = 0.001, batch_size = 327, num_epochs =
    # Set the model to training mode
    model.train()

    # Define the loss function and optimizer
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)

    # Initialize the lists for training information
    iters, losses, train_acc, val_acc = [], [], [], []

    # training
    n = 0 # the number of iterations

    start_time=time.time()
    for epoch in range(num_epochs):
        mini_b=0
        mini_batch_correct = 0
        Mini_batch_total = 0

        for imgs, labels in iter(train_loader):

            #Enable GPU
            if use_cuda and torch.cuda.is_available():
              imgs = imgs.cuda()
              labels = labels.cuda()

            # Forward pass
            if model_name == "alexnet":
              out = model(imgs)
              if n == 0:
                print("Using alexnet")
            elif model_name == 'resnet':
              if n == 0:
                print("Using resnet features")
            elif model_name == 'wide_resnet':
              if n == 0:
                print("Using wide_resnet features")
            else:
              if n == 0:
                print("Using efficientnet features")


            out = model(imgs)       # forward pass with Alexnet features
            loss = criterion(out, labels) # compute the loss
            loss.backward()               # backward pass
            optimizer.step()              # make update
            optimizer.zero_grad()         # clean up
```

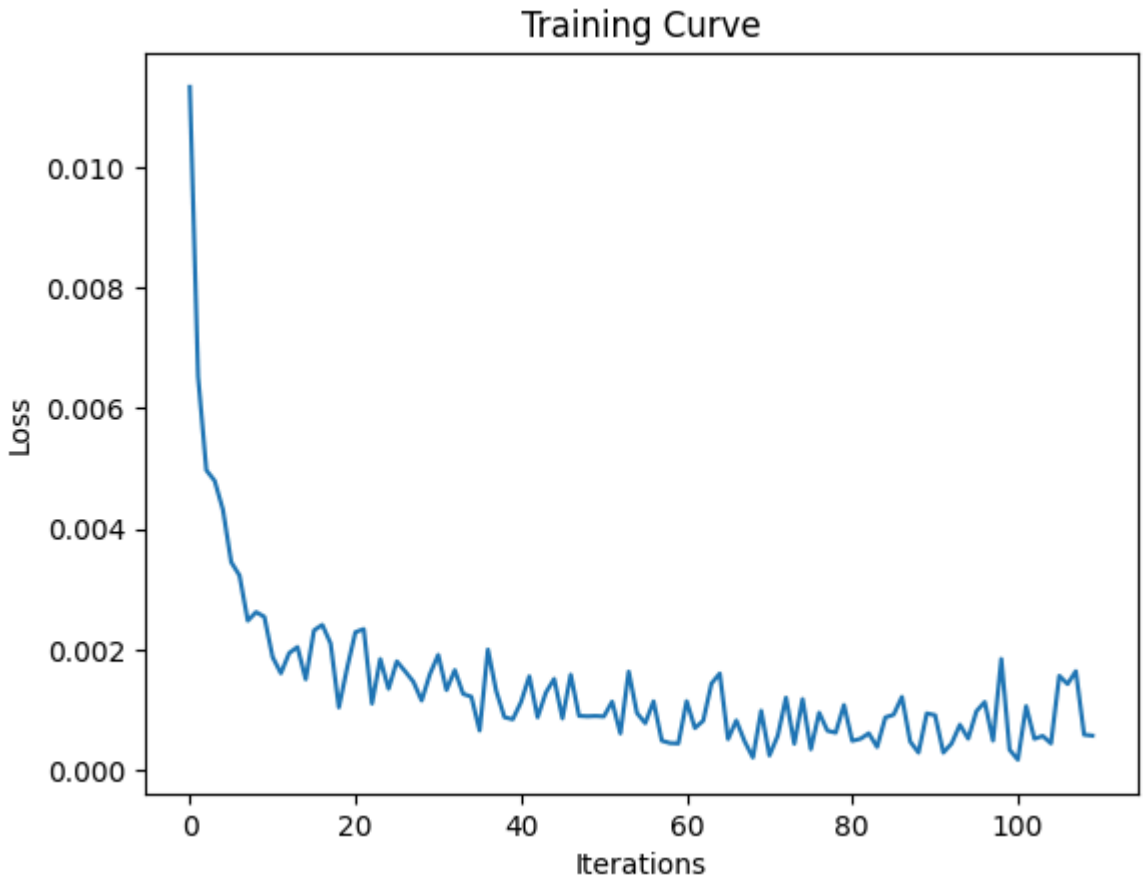
```
            # Get predicted class index
            _, pred = torch.max(out.data, 1) # Get predicted class index
            mini_batch_correct = pred.eq(labels.view_as(pred)).sum().item()
            Mini_batch_total = imgs.shape[0]
            train_acc.append((mini_batch_correct / Mini_batch_total))
            losses.append(float(loss)/batch_size)             # compute *average* l


            # Iterations
            iters.append(n)
            val_acc.append(get_accuracy_of_transfer(model, train=False))  # compute
            n += 1
            mini_b += 1
            print("Iteration: ",n,'Progress: % 6.2f ' % ((epoch * len(train_loader)


        print ("Epoch %d Finished. " % epoch ,"Time per Epoch: % 6.2f s "% ((time.t


    end_time= time.time()
    # Graphs
    plt.title("Training Curve")
    plt.plot(iters, losses, label="Train")
    plt.xlabel("Iterations")
    plt.ylabel("Loss")
    plt.show()

    plt.title("Training Curve")
    plt.plot(iters, train_acc, label="Training")
    plt.plot(iters, val_acc, label="Validation")
    plt.xlabel("Iterations")
    plt.ylabel("Validation Accuracy")
    plt.legend(loc='best')
    plt.show()

    train_acc.append(get_accuracy_of_transfer(model, train=True))
    print("Final Training Accuracy: {}".format(train_acc[-1]))
    print("Final Validation Accuracy: {}".format(val_acc[-1]))
    print ("Total time:  % 6.2f s  Time per Epoch: % 6.2f s " % ( (end_time-start_t
```

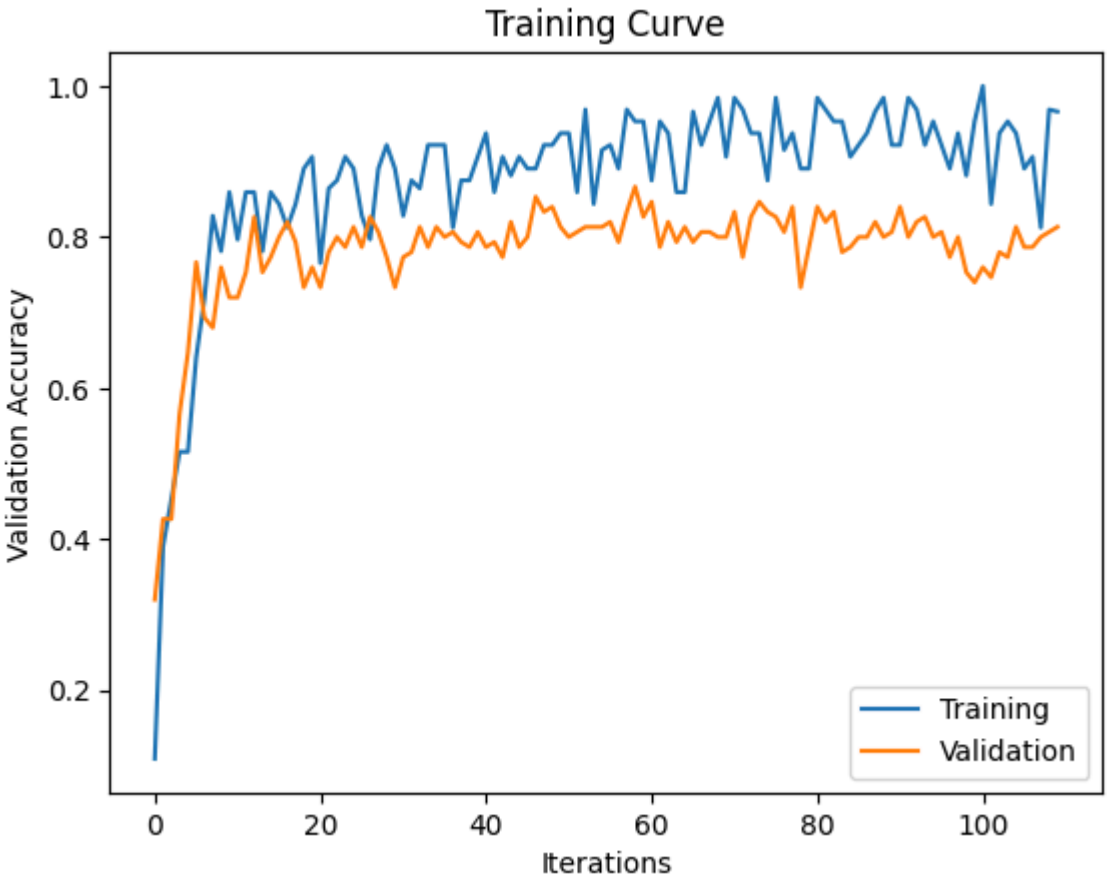
- Now we train the model

In [ ]:
```
use_cuda = True
torch.cuda.empty_cache()

model_alex = alexnet

model_name = "alexnet"
batch_size = 256

if use_cuda and torch.cuda.is_available():
  alexnet.cuda()
  model_alex.cuda()
  print('CUDA is available!  Training on GPU ...')
else:
  print('CUDA is not available.  Training on CPU ...')

# train model
train(model_alex, model_name, batch_size=batch_size, num_epochs=10)
```

```
CUDA is not available.  Training on CPU ...
Using alexnet
Iteration:  1 Progress:   0.91  % Time Elapsed:  28.95 s
Iteration:  2 Progress:   1.82  % Time Elapsed:  56.63 s
Iteration:  3 Progress:   2.73  % Time Elapsed:  83.12 s
Iteration:  4 Progress:   3.64  % Time Elapsed:  112.67 s
Iteration:  5 Progress:   4.55  % Time Elapsed:  138.74 s
Iteration:  6 Progress:   5.45  % Time Elapsed:  166.16 s
Iteration:  7 Progress:   6.36  % Time Elapsed:  193.84 s
Iteration:  8 Progress:   7.27  % Time Elapsed:  218.59 s
Iteration:  9 Progress:   8.18  % Time Elapsed:  244.88 s
Iteration:  10 Progress:   9.09  % Time Elapsed:  272.74 s
Iteration:  11 Progress:  10.00  % Time Elapsed:  298.05 s
Epoch 0 Finished.  Time per Epoch:  298.05 s
Iteration:  12 Progress:  10.91  % Time Elapsed:  309.97 s
Iteration:  13 Progress:  11.82  % Time Elapsed:  322.26 s
Iteration:  14 Progress:  12.73  % Time Elapsed:  334.46 s
Iteration:  15 Progress:  13.64  % Time Elapsed:  346.21 s
Iteration:  16 Progress:  14.55  % Time Elapsed:  357.34 s
Iteration:  17 Progress:  15.45  % Time Elapsed:  369.95 s
Iteration:  18 Progress:  16.36  % Time Elapsed:  382.18 s
Iteration:  19 Progress:  17.27  % Time Elapsed:  397.64 s
Iteration:  20 Progress:  18.18  % Time Elapsed:  409.69 s
Iteration:  21 Progress:  19.09  % Time Elapsed:  421.38 s
Iteration:  22 Progress:  20.00  % Time Elapsed:  433.66 s
Epoch 1 Finished.  Time per Epoch:  216.83 s
Iteration:  23 Progress:  20.91  % Time Elapsed:  446.17 s
Iteration:  24 Progress:  21.82  % Time Elapsed:  458.65 s
Iteration:  25 Progress:  22.73  % Time Elapsed:  469.63 s
Iteration:  26 Progress:  23.64  % Time Elapsed:  481.92 s
Iteration:  27 Progress:  24.55  % Time Elapsed:  494.27 s
Iteration:  28 Progress:  25.45  % Time Elapsed:  507.02 s
Iteration:  29 Progress:  26.36  % Time Elapsed:  518.93 s
Iteration:  30 Progress:  27.27  % Time Elapsed:  530.24 s
Iteration:  31 Progress:  28.18  % Time Elapsed:  542.57 s
Iteration:  32 Progress:  29.09  % Time Elapsed:  555.01 s
Iteration:  33 Progress:  30.00  % Time Elapsed:  567.32 s
Epoch 2 Finished.  Time per Epoch:  189.11 s
Iteration:  34 Progress:  30.91  % Time Elapsed:  578.72 s
Iteration:  35 Progress:  31.82  % Time Elapsed:  590.62 s
Iteration:  36 Progress:  32.73  % Time Elapsed:  603.24 s
Iteration:  37 Progress:  33.64  % Time Elapsed:  615.51 s
Iteration:  38 Progress:  34.55  % Time Elapsed:  628.21 s
Iteration:  39 Progress:  35.45  % Time Elapsed:  640.98 s
Iteration:  40 Progress:  36.36  % Time Elapsed:  652.49 s
Iteration:  41 Progress:  37.27  % Time Elapsed:  665.36 s
Iteration:  42 Progress:  38.18  % Time Elapsed:  678.03 s
Iteration:  43 Progress:  39.09  % Time Elapsed:  690.61 s
Iteration:  44 Progress:  40.00  % Time Elapsed:  701.50 s
Epoch 3 Finished.  Time per Epoch:  175.38 s
Iteration:  45 Progress:  40.91  % Time Elapsed:  713.41 s
Iteration:  46 Progress:  41.82  % Time Elapsed:  725.79 s
Iteration:  47 Progress:  42.73  % Time Elapsed:  738.01 s
Iteration:  48 Progress:  43.64  % Time Elapsed:  750.80 s
Iteration:  49 Progress:  44.55  % Time Elapsed:  761.73 s
Iteration:  50 Progress:  45.45  % Time Elapsed:  773.97 s
Iteration:  51 Progress:  46.36  % Time Elapsed:  786.48 s
Iteration:  52 Progress:  47.27  % Time Elapsed:  799.04 s
Iteration:  53 Progress:  48.18  % Time Elapsed:  811.28 s
Iteration:  54 Progress:  49.09  % Time Elapsed:  822.42 s
Iteration:  55 Progress:  50.00  % Time Elapsed:  834.57 s
Epoch 4 Finished.  Time per Epoch:  166.91 s
Iteration:  56 Progress:  50.91  % Time Elapsed:  847.05 s
Iteration:  57 Progress:  51.82  % Time Elapsed:  859.68 s
```

```
Iteration:  58 Progress:  52.73  % Time Elapsed:  871.65 s
Iteration:  59 Progress:  53.64  % Time Elapsed:  883.09 s
Iteration:  60 Progress:  54.55  % Time Elapsed:  895.95 s
Iteration:  61 Progress:  55.45  % Time Elapsed:  908.49 s
Iteration:  62 Progress:  56.36  % Time Elapsed:  921.09 s
Iteration:  63 Progress:  57.27  % Time Elapsed:  932.92 s
Iteration:  64 Progress:  58.18  % Time Elapsed:  944.61 s
Iteration:  65 Progress:  59.09  % Time Elapsed:  957.30 s
Iteration:  66 Progress:  60.00  % Time Elapsed:  969.54 s
Epoch 5 Finished.  Time per Epoch:  161.59 s
Iteration:  67 Progress:  60.91  % Time Elapsed:  981.87 s
Iteration:  68 Progress:  61.82  % Time Elapsed:  993.08 s
Iteration:  69 Progress:  62.73  % Time Elapsed:  1008.29 s
Iteration:  70 Progress:  63.64  % Time Elapsed:  1020.94 s
Iteration:  71 Progress:  64.55  % Time Elapsed:  1033.68 s
Iteration:  72 Progress:  65.45  % Time Elapsed:  1046.07 s
Iteration:  73 Progress:  66.36  % Time Elapsed:  1058.29 s
Iteration:  74 Progress:  67.27  % Time Elapsed:  1068.93 s
Iteration:  75 Progress:  68.18  % Time Elapsed:  1081.49 s
Iteration:  76 Progress:  69.09  % Time Elapsed:  1093.86 s
Iteration:  77 Progress:  70.00  % Time Elapsed:  1106.20 s
Epoch 6 Finished.  Time per Epoch:  158.03 s
Iteration:  78 Progress:  70.91  % Time Elapsed:  1118.01 s
Iteration:  79 Progress:  71.82  % Time Elapsed:  1129.29 s
Iteration:  80 Progress:  72.73  % Time Elapsed:  1141.76 s
Iteration:  81 Progress:  73.64  % Time Elapsed:  1154.25 s
Iteration:  82 Progress:  74.55  % Time Elapsed:  1166.69 s
Iteration:  83 Progress:  75.45  % Time Elapsed:  1178.59 s
Iteration:  84 Progress:  76.36  % Time Elapsed:  1189.65 s
Iteration:  85 Progress:  77.27  % Time Elapsed:  1202.65 s
Iteration:  86 Progress:  78.18  % Time Elapsed:  1214.99 s
Iteration:  87 Progress:  79.09  % Time Elapsed:  1227.61 s
Iteration:  88 Progress:  80.00  % Time Elapsed:  1239.21 s
Epoch 7 Finished.  Time per Epoch:  154.90 s
Iteration:  89 Progress:  80.91  % Time Elapsed:  1250.22 s
Iteration:  90 Progress:  81.82  % Time Elapsed:  1263.00 s
Iteration:  91 Progress:  82.73  % Time Elapsed:  1275.54 s
Iteration:  92 Progress:  83.64  % Time Elapsed:  1287.93 s
Iteration:  93 Progress:  84.55  % Time Elapsed:  1299.73 s
Iteration:  94 Progress:  85.45  % Time Elapsed:  1310.74 s
Iteration:  95 Progress:  86.36  % Time Elapsed:  1323.26 s
Iteration:  96 Progress:  87.27  % Time Elapsed:  1335.59 s
Iteration:  97 Progress:  88.18  % Time Elapsed:  1348.09 s
Iteration:  98 Progress:  89.09  % Time Elapsed:  1359.98 s
Iteration:  99 Progress:  90.00  % Time Elapsed:  1371.08 s
Epoch 8 Finished.  Time per Epoch:  152.34 s
Iteration:  100 Progress:  90.91  % Time Elapsed:  1383.73 s
Iteration:  101 Progress:  91.82  % Time Elapsed:  1396.18 s
Iteration:  102 Progress:  92.73  % Time Elapsed:  1408.73 s
Iteration:  103 Progress:  93.64  % Time Elapsed:  1420.20 s
Iteration:  104 Progress:  94.55  % Time Elapsed:  1431.69 s
Iteration:  105 Progress:  95.45  % Time Elapsed:  1444.31 s
Iteration:  106 Progress:  96.36  % Time Elapsed:  1456.79 s
Iteration:  107 Progress:  97.27  % Time Elapsed:  1469.10 s
Iteration:  108 Progress:  98.18  % Time Elapsed:  1481.09 s
Iteration:  109 Progress:  99.09  % Time Elapsed:  1492.53 s
Iteration:  110 Progress:  100.00  % Time Elapsed:  1504.77 s
Epoch 9 Finished.  Time per Epoch:  150.48 s
```






```
Final Training Accuracy: 0.9256080114449213
Final Validation Accuracy: 0.8133333333333334
Total time:   1504.77 s  Time per Epoch:  150.48 s
```

- Now we see how well our transfer learning model performs, we can proceed to using our large data set of 120 species and 12000 samples to see how well it performs.

- Here we redefine the path to our large data set

In [ ]:
```
master_path = '/content/gdrive/MyDrive/aps360largetest/Images'
img_width, img_height = 224, 224
channels = 3
batch_size = 64
num_images= 992
image_arr_size= img_width * img_height * channels
```

In [ ]:
```
cd /content/gdrive/MyDrive/aps360largetest/Images
```

```
/content/gdrive/.shortcut-targets-by-id/1fLRn_-T5O6uh9f4YOAyHJSmQT7pYg_6y/Images
```

In [ ]:
```
# Define the paths
image_folder = '.'  # Current directory since you're already in the image folder
train_folder = 'train'
val_folder = 'val'
test_folder = 'test'
```

In [ ]:
```
if first_time:

  print("Splitting Images into 70% training, 15% validation, 15% testing")
  for class_folder in os.listdir(small_test_folder):
      class_path = os.path.join(small_test_folder, class_folder)
      if os.path.isdir(class_path):  # Check if it's a directory (class folder)

          # Create corresponding class folders in train, val, test
          os.makedirs(os.path.join(small_test_train_folder, class_folder), exist_ok
          os.makedirs(os.path.join(small_test_val_folder, class_folder), exist_ok=T
          os.makedirs(os.path.join(small_test_test_folder, class_folder), exist_ok=

          # Get the list of image files for this class
          image_files = [f for f in os.listdir(class_path) if os.path.isfile(os.pat

          # Split into train and temp (test + val)
          train_files, temp_files = train_test_split(image_files, test_size=0.3, ra

          # Split temp into val and test
          val_files, test_files = train_test_split(temp_files, test_size=0.5, randc

          # Copy the files to their respective folders
          for file in train_files:
              shutil.copy(os.path.join(class_path, file), os.path.join(small_test_t
          for file in val_files:
              shutil.copy(os.path.join(class_path, file), os.path.join(small_test_v
          for file in test_files:
              shutil.copy(os.path.join(class_path, file), os.path.join(small_test_t

  first_time = False

else:
  print("Images Already Split")
```

```
Images Already Split
```

In [ ]:
```
first_time = True
```

In [ ]:
```
def count_images(folder):
    count = 0
    if first_time:
      # Define the paths
```

```
        train_folder = '/content/gdrive/MyDrive/aps360largetest/Images/train'
        val_folder = '/content/gdrive/MyDrive/aps360largetest/Images/val'
        test_folder = '/content/gdrive/MyDrive/aps360largetest/Images/test'

        for class_folder in os.listdir(folder):
            class_path = os.path.join(folder, class_folder)
            if os.path.isdir(class_path):
                count += len([f for f in os.listdir(class_path) if os.path.isfile(os.

    return count

train_count = count_images(train_folder)
val_count = count_images(val_folder)
test_count = count_images(test_folder)


total_count = train_count + val_count + test_count

if first_time:
  print("Number of training images:", train_count, "({:.2f}%)".format(train_count/t
  print("Number of validation images:", val_count, "({:.2f}%)".format(val_count/tot
  print("Number of testing images:", test_count, "({:.2f}%)".format(test_count/tota
else:
  print("Images Already counted")
```

```
Number of training images:14406 (70.00%)
Number of validation images:3087 (15.00%)
Number of testing images:3087 (15.00%)
```

In [ ]:
```
# Define data transformations
data_transform = transforms.Compose([
    transforms.Resize((224, 224)),  # Resize to match CNN input
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])  # Normalize us
])

# Load training data
train_dataset = datasets.ImageFolder(root='train', transform=data_transform)
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)

# Load validation data
val_dataset = datasets.ImageFolder(root='val', transform=data_transform)
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)

# Load test data
test_dataset = datasets.ImageFolder(root='test', transform=data_transform)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
```

In [ ]:
```
num_classes = 120
alexnet
```

In [ ]:
```
use_cuda = True
torch.cuda.empty_cache()

model_alex = alexnet

model_name = "alexnet"
batch_size = 327

if use_cuda and torch.cuda.is_available():
  alexnet.cuda()
  model_alex.cuda()
  print('CUDA is available!  Training on GPU ...')
else:
```

```
  print('CUDA is not available.  Training on CPU ...')

# train model
train(model_alex, model_name, batch_size=batch_size, num_epochs=10)
```

```
CUDA is available!  Training on GPU ...
Using alexnet
Iteration:  1 Progress:   0.04  % Time Elapsed:  2344.04 s
Iteration:  2 Progress:   0.09  % Time Elapsed:  2413.71 s
Iteration:  3 Progress:   0.13  % Time Elapsed:  2482.66 s
Iteration:  4 Progress:   0.18  % Time Elapsed:  2553.10 s
Iteration:  5 Progress:   0.22  % Time Elapsed:  2623.15 s
Iteration:  6 Progress:   0.27  % Time Elapsed:  2691.94 s
Iteration:  7 Progress:   0.31  % Time Elapsed:  2759.37 s
Iteration:  8 Progress:   0.36  % Time Elapsed:  2829.61 s
Iteration:  9 Progress:   0.40  % Time Elapsed:  2899.18 s
Iteration:  10 Progress:   0.44  % Time Elapsed:  2968.48 s
Iteration:  11 Progress:   0.49  % Time Elapsed:  3040.04 s
Iteration:  12 Progress:   0.53  % Time Elapsed:  3112.32 s
Iteration:  13 Progress:   0.58  % Time Elapsed:  3182.04 s
Iteration:  14 Progress:   0.62  % Time Elapsed:  3253.16 s
Iteration:  15 Progress:   0.67  % Time Elapsed:  3325.06 s
Iteration:  16 Progress:   0.71  % Time Elapsed:  3395.01 s
Iteration:  17 Progress:   0.76  % Time Elapsed:  3464.38 s
Iteration:  18 Progress:   0.80  % Time Elapsed:  3534.73 s
Iteration:  19 Progress:   0.84  % Time Elapsed:  3604.11 s
Iteration:  20 Progress:   0.89  % Time Elapsed:  3675.08 s
Iteration:  21 Progress:   0.93  % Time Elapsed:  3746.32 s
Iteration:  22 Progress:   0.98  % Time Elapsed:  3814.95 s
```

- From the abovee outcome we tried training the large transfer learning data set, however it was too time consuming. As we ran over 22 iterations we found the whole training process can take up to 50 hours even if we already applied our fastest avaliable gpu. This is probably because we had such a large data set of 120 classes and more than 12000 images to process. So for this specific task, we decided to cut our data size, for example, reduce to 20 classes and for each class, only take 100 images. Now we modify how we split the data below.

In [ ]:
```
cd /content/gdrive/MyDrive/aps360largetest/Images
```

```
/content/gdrive/.shortcut-targets-by-id/1fLRn_-T5O6uh9f4YOAyHJSmQT7pYg_6y/Images
```

In [ ]:
```
# Define the paths
large_test_folder = '.'  # Current directory
large_test_train_folder = 'train_2'
large_test_val_folder = 'val_2'
large_test_test_folder = 'test_2'
```

In [ ]:
```
large_test_folder = '/content/gdrive/MyDrive/aps360largetest/Images'
large_test_train_folder = '/content/gdrive/MyDrive/aps360largetest/Images/train_2'
large_test_val_folder = '/content/gdrive/MyDrive/aps360largetest/Images/val_2'
large_test_test_folder = '/content/gdrive/MyDrive/aps360largetest/Images/test_2'
```

In [ ]:
```
first_time = True
num_images_per_class = 100
num_classes = 20
number_of_selected_class = 20
batch_size = 327
```

In [ ]:
```
# Re split the data in another folder
if first_time:

  # Get all class folders
  all_classes = [f for f in os.listdir(large_test_folder) if os.path.isdir(os.path.

  # Randomly select 20 classes
  selected_classes = random.sample(all_classes, number_of_selected_class)

  print("Splitting Images into 70% training, 15% validation, 15% testing")
  for class_folder in selected_classes:
      class_path = os.path.join(large_test_folder, class_folder)

      # Create corresponding class folders in train, val, test
      os.makedirs(os.path.join(large_test_train_folder, class_folder), exist_ok=Tru
      os.makedirs(os.path.join(large_test_val_folder, class_folder), exist_ok=True)
      os.makedirs(os.path.join(large_test_test_folder, class_folder), exist_ok=True

      # Get the list of image files for this class
      image_files = [f for f in os.listdir(class_path) if os.path.isfile(os.path.jo

      # Select up to 100 random images from the class
      num_images_to_select = min(num_images_per_class, len(image_files))
      selected_images = random.sample(image_files, num_images_to_select)

      # Split into train and temp (test + val)
      train_files, temp_files = train_test_split(selected_images, test_size=0.3, ra

      # Split temp into val and test
      val_files, test_files = train_test_split(temp_files, test_size=0.5, random_st

      # Copy the files to their respective folders
      for file in train_files:
          shutil.copy(os.path.join(class_path, file), os.path.join(large_test_train
      for file in val_files:
          shutil.copy(os.path.join(class_path, file), os.path.join(large_test_val_f
      for file in test_files:
          shutil.copy(os.path.join(class_path, file), os.path.join(large_test_test_

  first_time = False

else:
  print("Images Already Split")
```

```
Splitting Images into 70% training, 15% validation, 15% testing
```

In [ ]:
```
# Now we count to check if the data is correct
def count_images(folder):
    count = 0
    for class_folder in os.listdir(folder):
        class_path = os.path.join(folder, class_folder)
        if os.path.isdir(class_path):
            count += len([f for f in os.listdir(class_path) if os.path.isfile(os.pa
    return count

train_count = count_images(large_test_train_folder)
val_count = count_images(large_test_val_folder)
test_count = count_images(large_test_test_folder)


total_count = train_count + val_count + test_count

print("Number of training images:", train_count, "({:.2f}%)".format(train_count/tot
```

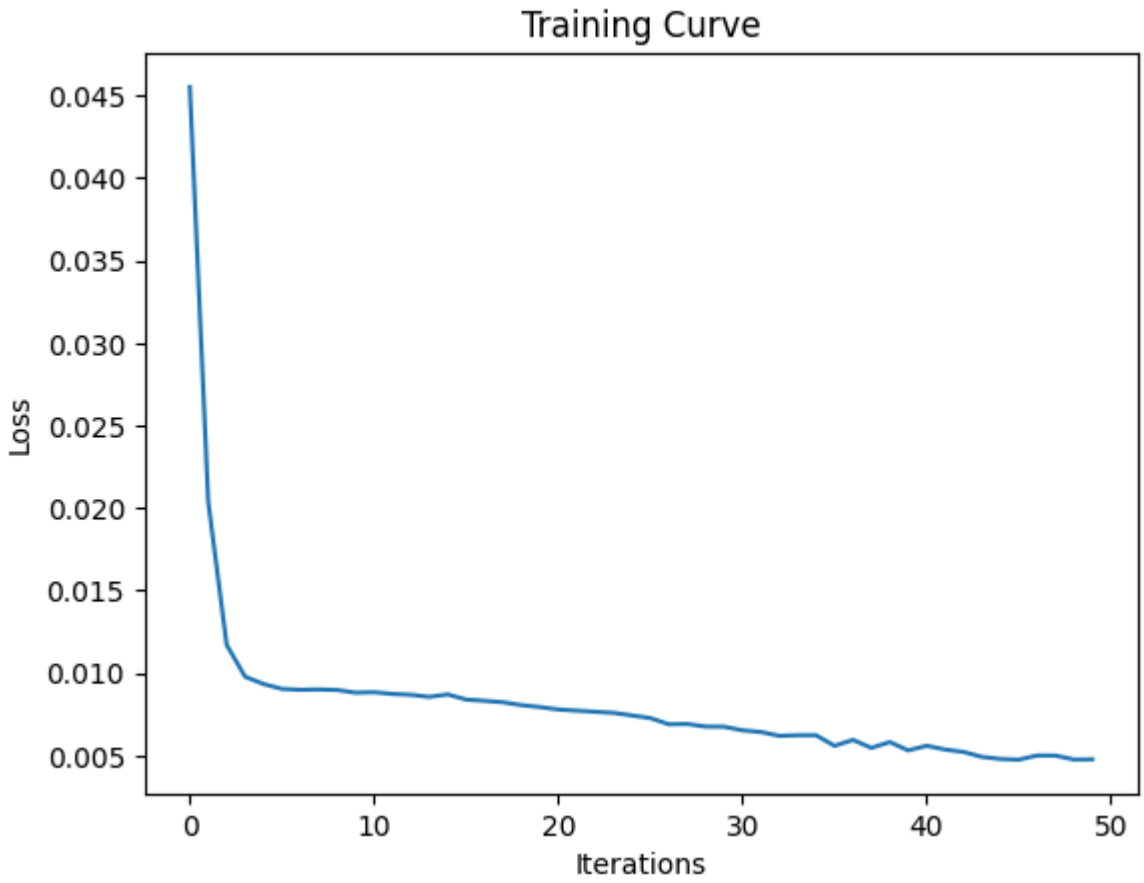
```
print("Number of validation images:", val_count, "({:.2f}%)".format(val_count/total
print("Number of testing images:", test_count, "({:.2f}%)".format(test_count/total_
```

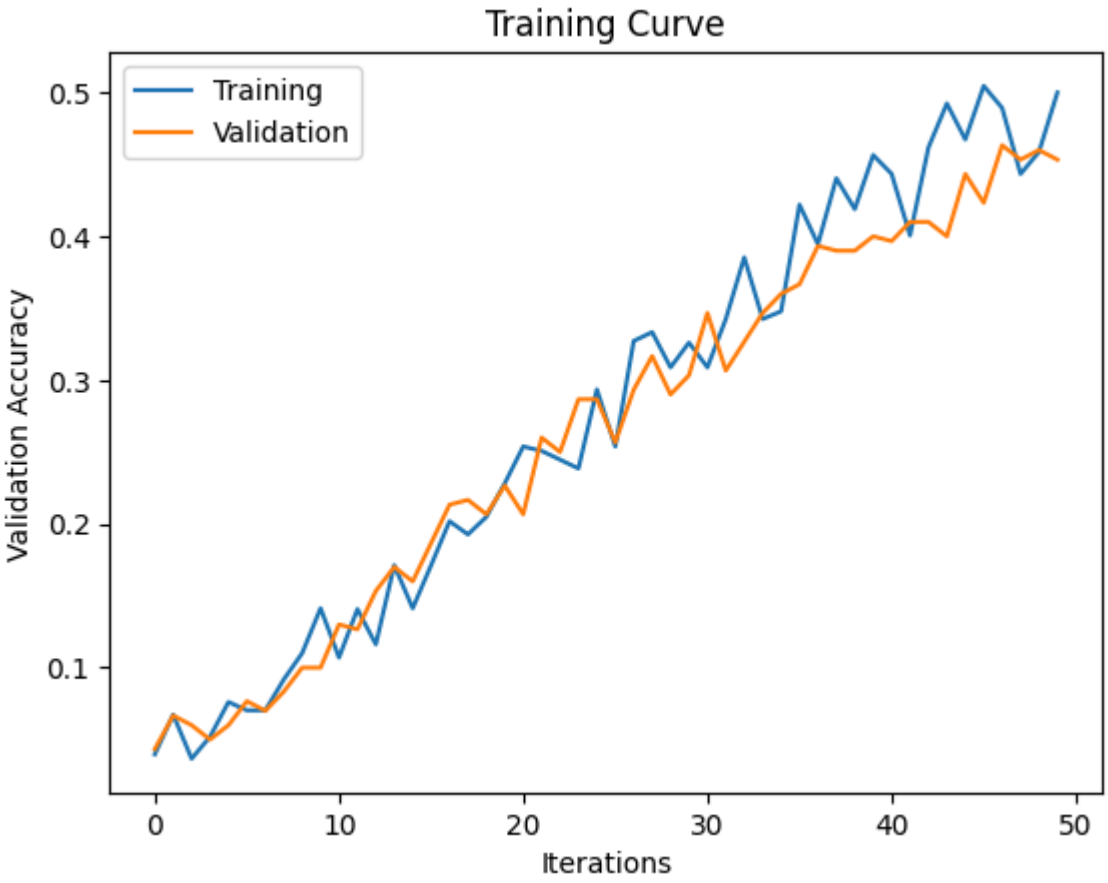
```
Number of training images: 1400 (70.00%)
Number of validation images: 300 (15.00%)
Number of testing images: 300 (15.00%)
```

In [ ]:
```
# Define data transformations
data_transform = transforms.Compose([
    transforms.Resize((224, 224)),  # Resize to match CNN input
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])  # Normalize us
])

# Load training data
train_dataset = datasets.ImageFolder(root='train_2', transform=data_transform)
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)

# Load validation data
val_dataset = datasets.ImageFolder(root='val_2', transform=data_transform)
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)

# Load test data
test_dataset = datasets.ImageFolder(root='test_2', transform=data_transform)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
```

In [ ]:
```
use_cuda = True
torch.cuda.empty_cache()

model_alex = alexnet

model_name = "alexnet"
batch_size = 327

if use_cuda and torch.cuda.is_available():
  alexnet.cuda()
  model_alex.cuda()
  print('CUDA is available!  Training on GPU ...')
else:
  print('CUDA is not available.  Training on CPU ...')

# train model
train(model_alex, model_name, batch_size=batch_size, num_epochs=10)
```

```
CUDA is not available.  Training on CPU ...
Using alexnet
Iteration:  1 Progress:   2.00  % Time Elapsed:  219.32 s
Iteration:  2 Progress:   4.00  % Time Elapsed:  349.82 s
Iteration:  3 Progress:   6.00  % Time Elapsed:  482.91 s
Iteration:  4 Progress:   8.00  % Time Elapsed:  610.40 s
Iteration:  5 Progress:  10.00  % Time Elapsed:  657.79 s
Epoch 0 Finished.  Time per Epoch:  657.79 s
Iteration:  6 Progress:  12.00  % Time Elapsed:  698.74 s
Iteration:  7 Progress:  14.00  % Time Elapsed:  739.99 s
Iteration:  8 Progress:  16.00  % Time Elapsed:  782.34 s
Iteration:  9 Progress:  18.00  % Time Elapsed:  824.86 s
Iteration:  10 Progress:  20.00  % Time Elapsed:  845.61 s
Epoch 1 Finished.  Time per Epoch:  422.81 s
Iteration:  11 Progress:  22.00  % Time Elapsed:  888.49 s
Iteration:  12 Progress:  24.00  % Time Elapsed:  930.89 s
Iteration:  13 Progress:  26.00  % Time Elapsed:  972.82 s
Iteration:  14 Progress:  28.00  % Time Elapsed:  1013.04 s
Iteration:  15 Progress:  30.00  % Time Elapsed:  1034.87 s
Epoch 2 Finished.  Time per Epoch:  344.96 s
Iteration:  16 Progress:  32.00  % Time Elapsed:  1077.23 s
Iteration:  17 Progress:  34.00  % Time Elapsed:  1118.84 s
Iteration:  18 Progress:  36.00  % Time Elapsed:  1158.59 s
Iteration:  19 Progress:  38.00  % Time Elapsed:  1199.29 s
Iteration:  20 Progress:  40.00  % Time Elapsed:  1221.97 s
Epoch 3 Finished.  Time per Epoch:  305.49 s
Iteration:  21 Progress:  42.00  % Time Elapsed:  1261.91 s
Iteration:  22 Progress:  44.00  % Time Elapsed:  1302.80 s
Iteration:  23 Progress:  46.00  % Time Elapsed:  1344.79 s
Iteration:  24 Progress:  48.00  % Time Elapsed:  1386.25 s
Iteration:  25 Progress:  50.00  % Time Elapsed:  1406.77 s
Epoch 4 Finished.  Time per Epoch:  281.35 s
Iteration:  26 Progress:  52.00  % Time Elapsed:  1448.64 s
Iteration:  27 Progress:  54.00  % Time Elapsed:  1491.00 s
Iteration:  28 Progress:  56.00  % Time Elapsed:  1538.37 s
Iteration:  29 Progress:  58.00  % Time Elapsed:  1580.61 s
Iteration:  30 Progress:  60.00  % Time Elapsed:  1601.23 s
Epoch 5 Finished.  Time per Epoch:  266.87 s
Iteration:  31 Progress:  62.00  % Time Elapsed:  1642.75 s
Iteration:  32 Progress:  64.00  % Time Elapsed:  1685.61 s
Iteration:  33 Progress:  66.00  % Time Elapsed:  1728.45 s
Iteration:  34 Progress:  68.00  % Time Elapsed:  1770.78 s
Iteration:  35 Progress:  70.00  % Time Elapsed:  1791.92 s
Epoch 6 Finished.  Time per Epoch:  255.99 s
Iteration:  36 Progress:  72.00  % Time Elapsed:  1834.43 s
Iteration:  37 Progress:  74.00  % Time Elapsed:  1877.04 s
Iteration:  38 Progress:  76.00  % Time Elapsed:  1922.28 s
Iteration:  39 Progress:  78.00  % Time Elapsed:  1967.90 s
Iteration:  40 Progress:  80.00  % Time Elapsed:  1989.61 s
Epoch 7 Finished.  Time per Epoch:  248.70 s
Iteration:  41 Progress:  82.00  % Time Elapsed:  2031.70 s
Iteration:  42 Progress:  84.00  % Time Elapsed:  2073.59 s
Iteration:  43 Progress:  86.00  % Time Elapsed:  2116.16 s
Iteration:  44 Progress:  88.00  % Time Elapsed:  2158.67 s
Iteration:  45 Progress:  90.00  % Time Elapsed:  2179.28 s
Epoch 8 Finished.  Time per Epoch:  242.14 s
Iteration:  46 Progress:  92.00  % Time Elapsed:  2221.12 s
Iteration:  47 Progress:  94.00  % Time Elapsed:  2263.41 s
Iteration:  48 Progress:  96.00  % Time Elapsed:  2306.33 s
Iteration:  49 Progress:  98.00  % Time Elapsed:  2348.40 s
Iteration:  50 Progress:  100.00  % Time Elapsed:  2369.71 s
Epoch 9 Finished.  Time per Epoch:  236.97 s
```






```
Final Training Accuracy: 0.5314285714285715
Final Validation Accuracy: 0.4533333333333333
Total time:   2369.72 s  Time per Epoch:  236.97 s
```

- From this testing we see if we reduced it to 20 classes and 100 images per class the training time was largely reduced. So next I am going to try with 50 classes.

In [ ]:
```
cd /content/gdrive/MyDrive/aps360largetest/Images
```

```
/content/gdrive/.shortcut-targets-by-id/1fLRn_-T5O6uh9f4YOAyHJSmQT7pYg_6y/Images
```

In [ ]:
```
# Define the paths
large_test_folder = '.'  # Current directory
large_test_train_folder = 'train_3'
large_test_val_folder = 'val_3'
large_test_test_folder = 'test_3'
```

In [ ]:
```
large_test_folder = '/content/gdrive/MyDrive/aps360largetest/Images'
large_test_train_folder = '/content/gdrive/MyDrive/aps360largetest/Images/train_3'
large_test_val_folder = '/content/gdrive/MyDrive/aps360largetest/Images/val_3'
large_test_test_folder = '/content/gdrive/MyDrive/aps360largetest/Images/test_3'
```

In [ ]:
```
first_time = True
num_images_per_class = 100
num_classes = 50
number_of_selected_class = 50
batch_size = 327
```

In [ ]:
```
first_time = False
```

In [ ]:
```
# Re split the data in another folder
if first_time:

  # Get all class folders, excluding "train", "valid", "test", "train_2", "valid_2"
  all_classes = [f for f in os.listdir(large_test_folder) if os.path.isdir(os.path.

  # Randomly select 50 classes
  selected_classes = random.sample(all_classes, number_of_selected_class)

  print("Splitting Images into 70% training, 15% validation, 15% testing")
  for class_folder in selected_classes:
      class_path = os.path.join(large_test_folder, class_folder)

      # Create corresponding class folders in train, val, test
      os.makedirs(os.path.join(large_test_train_folder, class_folder), exist_ok=Tru
      os.makedirs(os.path.join(large_test_val_folder, class_folder), exist_ok=True)
      os.makedirs(os.path.join(large_test_test_folder, class_folder), exist_ok=True

      # Get the list of image files for this class
      image_files = [f for f in os.listdir(class_path) if os.path.isfile(os.path.jo

      # Select up to 100 random images from the class
      num_images_to_select = min(num_images_per_class, len(image_files))
      selected_images = random.sample(image_files, num_images_to_select)

      # Split into train and temp (test + val)
      train_files, temp_files = train_test_split(selected_images, test_size=0.3, ra

      # Split temp into val and test
      val_files, test_files = train_test_split(temp_files, test_size=0.5, random_st

      # Copy the files to their respective folders
      for file in train_files:
          shutil.copy(os.path.join(class_path, file), os.path.join(large_test_train
      for file in val_files:
          shutil.copy(os.path.join(class_path, file), os.path.join(large_test_val_f
      for file in test_files:
          shutil.copy(os.path.join(class_path, file), os.path.join(large_test_test_
```

```
  first_time = False

else:
  print("Images Already Split")
```

```
Splitting Images into 70% training, 15% validation, 15% testing
```

In [ ]:
```
# Now we count to check if the data is correct
def count_images(folder):
    count = 0
    for class_folder in os.listdir(folder):
        class_path = os.path.join(folder, class_folder)
        if os.path.isdir(class_path):
            count += len([f for f in os.listdir(class_path) if os.path.isfile(os.pa
    return count

train_count = count_images(large_test_train_folder)
val_count = count_images(large_test_val_folder)
test_count = count_images(large_test_test_folder)


total_count = train_count + val_count + test_count

print("Number of training images:", train_count, "({:.2f}%)".format(train_count/tot
print("Number of validation images:", val_count, "({:.2f}%)".format(val_count/total
print("Number of testing images:", test_count, "({:.2f}%)".format(test_count/total_
```

```
Number of training images: 3500 (70.00%)
Number of validation images: 750 (15.00%)
Number of testing images: 750 (15.00%)
```

In [ ]:
```
# Define data transformations
data_transform = transforms.Compose([
    transforms.Resize((224, 224)),  # Resize to match CNN input
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])  # Normalize us
])

# Load training data
train_dataset = datasets.ImageFolder(root='train_3', transform=data_transform)
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)

# Load validation data
val_dataset = datasets.ImageFolder(root='val_3', transform=data_transform)
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)

# Load test data
test_dataset = datasets.ImageFolder(root='test_3', transform=data_transform)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
```

In [ ]:
```
num_classes = 50

alexnet = models.alexnet(pretrained=True)

#freeze model parameters, those pretrained alexnet parameters won't be updated duri
for param in alexnet.parameters():
  param.requires_grad = False

# We are adding final layers to AlexNet to fit our tasks for Transfer Learning
# In this case, since we are using small net data with only 5 classes,
# we are using small hidden layer size of 512 and drop out rate
# of 0.5 to make the model more accurate

# The structre of the modyfied alexnet is refernced to this following article:
```

```
# https://github.com/kipkoechjosh/APS360-Dog-Breed-Classifier/blob/main/final_dogs_
# After careful tunning of the neruals of each layer we decided to keep the followi

alexnet.classifier.add_module("4", nn.Linear(4096, 512))
alexnet.classifier.add_module("6", nn.Dropout(p=0.5, inplace=False))
alexnet.classifier.add_module("7", nn.Linear(512, 256))
alexnet.classifier.add_module("8", nn.ReLU(inplace=True))
alexnet.classifier.add_module("9", nn.Linear(256, num_classes))
alexnet
```

Out[ ]:

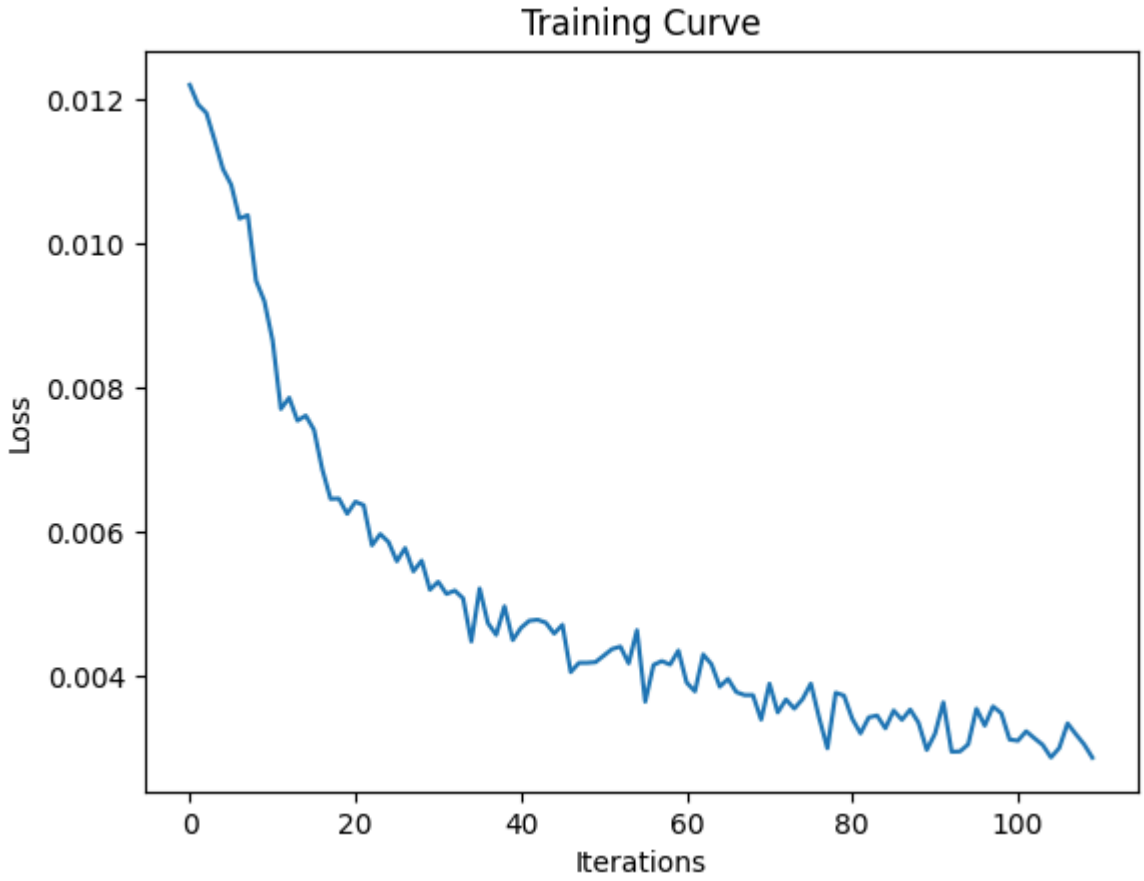
```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=Fals
e)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=Fals
e)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=Fals
e)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=512, bias=True)
    (5): ReLU(inplace=True)
    (6): Dropout(p=0.5, inplace=False)
    (7): Linear(in_features=512, out_features=256, bias=True)
    (8): ReLU(inplace=True)
    (9): Linear(in_features=256, out_features=50, bias=True)
  )
)
```

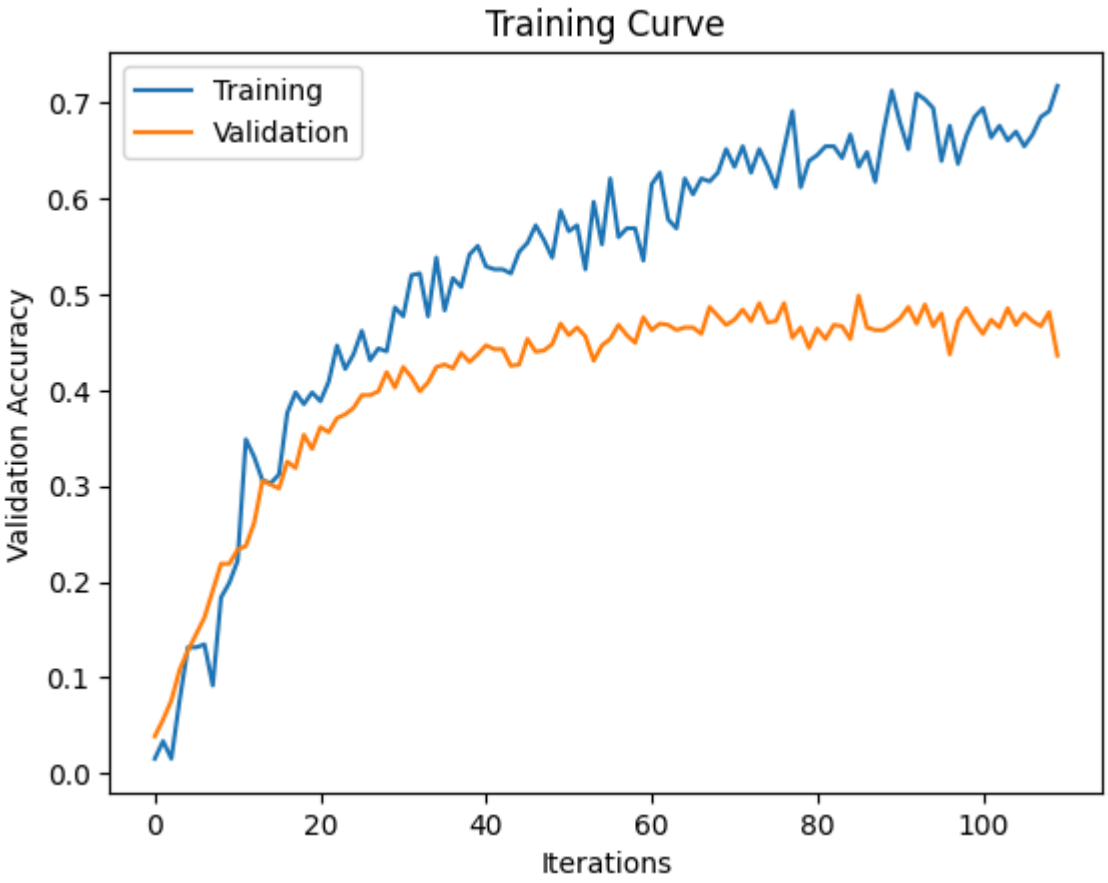
In [ ]:

```
use_cuda = True
torch.cuda.empty_cache()

model_alex = alexnet

model_name = "alexnet"


if use_cuda and torch.cuda.is_available():
  alexnet.cuda()
  model_alex.cuda()
  print('CUDA is available!  Training on GPU ...')
else:
  print('CUDA is not available.  Training on CPU ...')

# train model
train(model_alex, model_name, batch_size=batch_size, num_epochs=10)
```

```
CUDA is not available.  Training on CPU ...
Using alexnet
Iteration:  1 Progress:   0.91  % Time Elapsed:  94.94 s
Iteration:  2 Progress:   1.82  % Time Elapsed:  165.41 s
Iteration:  3 Progress:   2.73  % Time Elapsed:  230.51 s
Iteration:  4 Progress:   3.64  % Time Elapsed:  294.66 s
Iteration:  5 Progress:   4.55  % Time Elapsed:  358.93 s
Iteration:  6 Progress:   5.45  % Time Elapsed:  423.29 s
Iteration:  7 Progress:   6.36  % Time Elapsed:  485.90 s
Iteration:  8 Progress:   7.27  % Time Elapsed:  554.27 s
Iteration:  9 Progress:   8.18  % Time Elapsed:  630.43 s
Iteration:  10 Progress:   9.09  % Time Elapsed:  700.15 s
Iteration:  11 Progress:  10.00  % Time Elapsed:  755.35 s
Epoch 0 Finished.  Time per Epoch:  755.35 s
Iteration:  12 Progress:  10.91  % Time Elapsed:  817.99 s
Iteration:  13 Progress:  11.82  % Time Elapsed:  898.60 s
Iteration:  14 Progress:  12.73  % Time Elapsed:  979.42 s
Iteration:  15 Progress:  13.64  % Time Elapsed:  1051.39 s
Iteration:  16 Progress:  14.55  % Time Elapsed:  1118.75 s
Iteration:  17 Progress:  15.45  % Time Elapsed:  1185.10 s
Iteration:  18 Progress:  16.36  % Time Elapsed:  1256.28 s
Iteration:  19 Progress:  17.27  % Time Elapsed:  1325.36 s
Iteration:  20 Progress:  18.18  % Time Elapsed:  1393.82 s
Iteration:  21 Progress:  19.09  % Time Elapsed:  1461.24 s
Iteration:  22 Progress:  20.00  % Time Elapsed:  1520.19 s
Epoch 1 Finished.  Time per Epoch:  760.09 s
Iteration:  23 Progress:  20.91  % Time Elapsed:  1588.48 s
Iteration:  24 Progress:  21.82  % Time Elapsed:  1654.47 s
Iteration:  25 Progress:  22.73  % Time Elapsed:  1721.96 s
Iteration:  26 Progress:  23.64  % Time Elapsed:  1790.48 s
Iteration:  27 Progress:  24.55  % Time Elapsed:  1863.08 s
Iteration:  28 Progress:  25.45  % Time Elapsed:  1931.42 s
Iteration:  29 Progress:  26.36  % Time Elapsed:  1997.89 s
Iteration:  30 Progress:  27.27  % Time Elapsed:  2065.01 s
Iteration:  31 Progress:  28.18  % Time Elapsed:  2132.04 s
Iteration:  32 Progress:  29.09  % Time Elapsed:  2199.11 s
Iteration:  33 Progress:  30.00  % Time Elapsed:  2256.38 s
Epoch 2 Finished.  Time per Epoch:  752.13 s
Iteration:  34 Progress:  30.91  % Time Elapsed:  2322.78 s
Iteration:  35 Progress:  31.82  % Time Elapsed:  2389.86 s
Iteration:  36 Progress:  32.73  % Time Elapsed:  2457.28 s
Iteration:  37 Progress:  33.64  % Time Elapsed:  2525.47 s
Iteration:  38 Progress:  34.55  % Time Elapsed:  2591.93 s
Iteration:  39 Progress:  35.45  % Time Elapsed:  2666.82 s
Iteration:  40 Progress:  36.36  % Time Elapsed:  2733.99 s
Iteration:  41 Progress:  37.27  % Time Elapsed:  2800.97 s
Iteration:  42 Progress:  38.18  % Time Elapsed:  2866.77 s
Iteration:  43 Progress:  39.09  % Time Elapsed:  2933.74 s
Iteration:  44 Progress:  40.00  % Time Elapsed:  2992.74 s
Epoch 3 Finished.  Time per Epoch:  748.19 s
Iteration:  45 Progress:  40.91  % Time Elapsed:  3060.13 s
Iteration:  46 Progress:  41.82  % Time Elapsed:  3126.44 s
Iteration:  47 Progress:  42.73  % Time Elapsed:  3193.16 s
Iteration:  48 Progress:  43.64  % Time Elapsed:  3259.50 s
Iteration:  49 Progress:  44.55  % Time Elapsed:  3326.87 s
Iteration:  50 Progress:  45.45  % Time Elapsed:  3393.94 s
Iteration:  51 Progress:  46.36  % Time Elapsed:  3461.35 s
Iteration:  52 Progress:  47.27  % Time Elapsed:  3529.57 s
Iteration:  53 Progress:  48.18  % Time Elapsed:  3596.94 s
Iteration:  54 Progress:  49.09  % Time Elapsed:  3664.49 s
Iteration:  55 Progress:  50.00  % Time Elapsed:  3722.01 s
Epoch 4 Finished.  Time per Epoch:  744.40 s
Iteration:  56 Progress:  50.91  % Time Elapsed:  3788.51 s
Iteration:  57 Progress:  51.82  % Time Elapsed:  3855.98 s
```

```
Iteration:  58 Progress:  52.73  % Time Elapsed:  3924.20 s
Iteration:  59 Progress:  53.64  % Time Elapsed:  3992.00 s
Iteration:  60 Progress:  54.55  % Time Elapsed:  4059.18 s
Iteration:  61 Progress:  55.45  % Time Elapsed:  4125.23 s
Iteration:  62 Progress:  56.36  % Time Elapsed:  4191.77 s
Iteration:  63 Progress:  57.27  % Time Elapsed:  4258.45 s
Iteration:  64 Progress:  58.18  % Time Elapsed:  4325.95 s
Iteration:  65 Progress:  59.09  % Time Elapsed:  4392.83 s
Iteration:  66 Progress:  60.00  % Time Elapsed:  4450.66 s
Epoch 5 Finished.  Time per Epoch:  741.78 s
Iteration:  67 Progress:  60.91  % Time Elapsed:  4517.88 s
Iteration:  68 Progress:  61.82  % Time Elapsed:  4585.02 s
Iteration:  69 Progress:  62.73  % Time Elapsed:  4652.24 s
Iteration:  70 Progress:  63.64  % Time Elapsed:  4718.13 s
Iteration:  71 Progress:  64.55  % Time Elapsed:  4784.60 s
Iteration:  72 Progress:  65.45  % Time Elapsed:  4850.48 s
Iteration:  73 Progress:  66.36  % Time Elapsed:  4919.63 s
Iteration:  74 Progress:  67.27  % Time Elapsed:  4987.53 s
Iteration:  75 Progress:  68.18  % Time Elapsed:  5056.52 s
Iteration:  76 Progress:  69.09  % Time Elapsed:  5123.26 s
Iteration:  77 Progress:  70.00  % Time Elapsed:  5183.16 s
Epoch 6 Finished.  Time per Epoch:  740.45 s
Iteration:  78 Progress:  70.91  % Time Elapsed:  5250.10 s
Iteration:  79 Progress:  71.82  % Time Elapsed:  5318.38 s
Iteration:  80 Progress:  72.73  % Time Elapsed:  5385.74 s
Iteration:  81 Progress:  73.64  % Time Elapsed:  5453.34 s
Iteration:  82 Progress:  74.55  % Time Elapsed:  5520.35 s
Iteration:  83 Progress:  75.45  % Time Elapsed:  5589.35 s
Iteration:  84 Progress:  76.36  % Time Elapsed:  5656.43 s
Iteration:  85 Progress:  77.27  % Time Elapsed:  5725.07 s
Iteration:  86 Progress:  78.18  % Time Elapsed:  5792.37 s
Iteration:  87 Progress:  79.09  % Time Elapsed:  5861.25 s
Iteration:  88 Progress:  80.00  % Time Elapsed:  5917.91 s
Epoch 7 Finished.  Time per Epoch:  739.74 s
Iteration:  89 Progress:  80.91  % Time Elapsed:  5986.46 s
Iteration:  90 Progress:  81.82  % Time Elapsed:  6053.00 s
Iteration:  91 Progress:  82.73  % Time Elapsed:  6121.67 s
Iteration:  92 Progress:  83.64  % Time Elapsed:  6189.32 s
Iteration:  93 Progress:  84.55  % Time Elapsed:  6257.25 s
Iteration:  94 Progress:  85.45  % Time Elapsed:  6324.11 s
Iteration:  95 Progress:  86.36  % Time Elapsed:  6391.97 s
Iteration:  96 Progress:  87.27  % Time Elapsed:  6458.58 s
Iteration:  97 Progress:  88.18  % Time Elapsed:  6525.95 s
Iteration:  98 Progress:  89.09  % Time Elapsed:  6593.83 s
Iteration:  99 Progress:  90.00  % Time Elapsed:  6653.18 s
Epoch 8 Finished.  Time per Epoch:  739.24 s
Iteration:  100 Progress:  90.91  % Time Elapsed:  6722.85 s
Iteration:  101 Progress:  91.82  % Time Elapsed:  6791.02 s
Iteration:  102 Progress:  92.73  % Time Elapsed:  6860.94 s
Iteration:  103 Progress:  93.64  % Time Elapsed:  6928.83 s
Iteration:  104 Progress:  94.55  % Time Elapsed:  6998.94 s
Iteration:  105 Progress:  95.45  % Time Elapsed:  7067.45 s
Iteration:  106 Progress:  96.36  % Time Elapsed:  7136.67 s
Iteration:  107 Progress:  97.27  % Time Elapsed:  7202.64 s
Iteration:  108 Progress:  98.18  % Time Elapsed:  7265.08 s
Iteration:  109 Progress:  99.09  % Time Elapsed:  7326.23 s
Iteration:  110 Progress:  100.00  % Time Elapsed:  7379.85 s
Epoch 9 Finished.  Time per Epoch:  737.99 s
```






```
Final Training Accuracy: 0.7154285714285714
Final Validation Accuracy: 0.436
Total time:   7379.85 s  Time per Epoch:  737.99 s
```

- from the above, using data set of 50 classes and 100 samples per class gave us a result of 900 seconds training time. This is approximitely 15 minutes. For the sake of time, we

will cap our model at this point and no longer increase data size.

- The task now is trying to find the hyperparameters to make the model better.

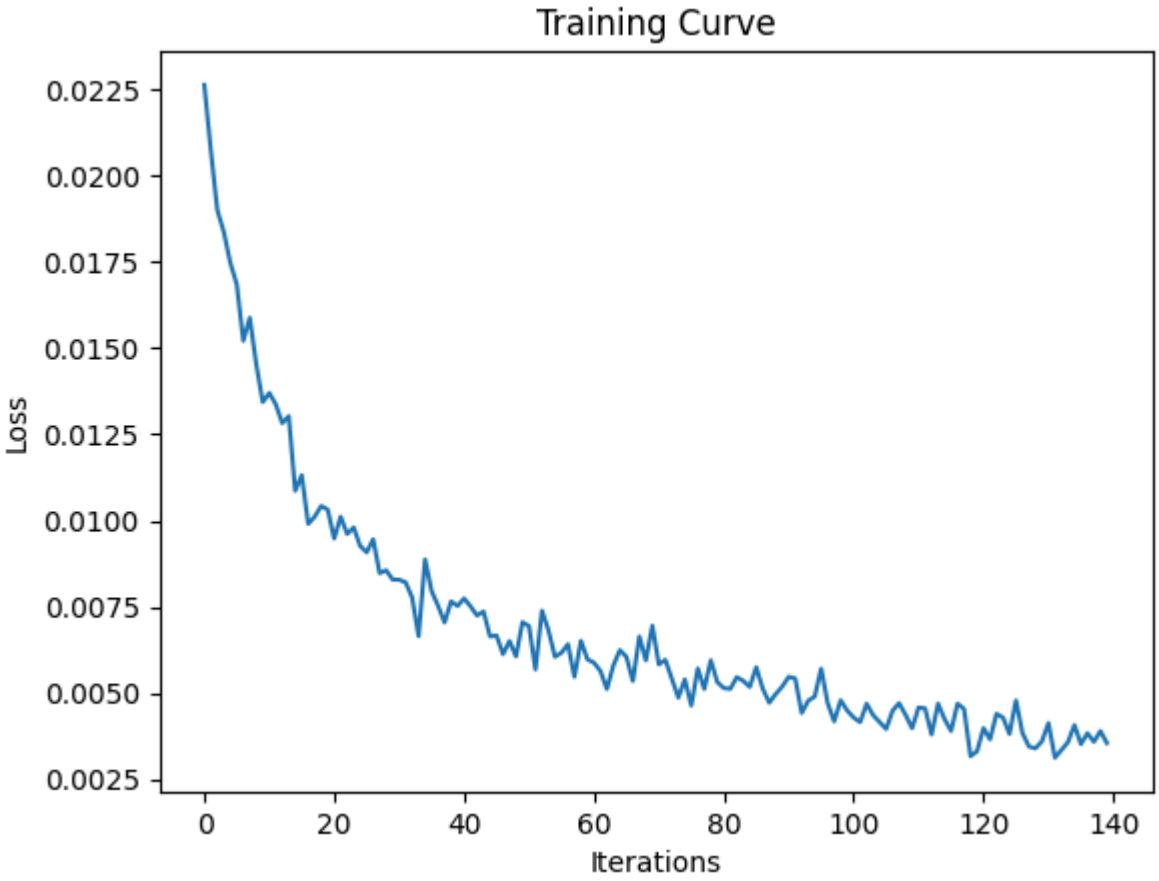
- In Here the training time was slower because at first we used GPU training. However later we exceeded our colab quota and was not able to use gpu anymore. So instead we changed back to CPU

In [ ]:
```
#Trying Batch size 256
use_cuda = True
torch.cuda.empty_cache()
batch_size = 256
model_alex = alexnet

model_name = "alexnet"


if use_cuda and torch.cuda.is_available():
  alexnet.cuda()
  model_alex.cuda()
  print('CUDA is available!  Training on GPU ...')
else:
  print('CUDA is not available.  Training on CPU ...')

# train model
train(model_alex, model_name, batch_size=batch_size, num_epochs=10)
```

```
CUDA is available!  Training on GPU ...
Using alexnet
Iteration:  1 Progress:   0.71  % Time Elapsed:  324.96 s
Iteration:  2 Progress:   1.43  % Time Elapsed:  415.73 s
Iteration:  3 Progress:   2.14  % Time Elapsed:  510.18 s
Iteration:  4 Progress:   2.86  % Time Elapsed:  606.89 s
Iteration:  5 Progress:   3.57  % Time Elapsed:  700.26 s
Iteration:  6 Progress:   4.29  % Time Elapsed:  797.00 s
Iteration:  7 Progress:   5.00  % Time Elapsed:  893.47 s
Iteration:  8 Progress:   5.71  % Time Elapsed:  989.81 s
Iteration:  9 Progress:   6.43  % Time Elapsed:  1084.89 s
Iteration:  10 Progress:   7.14  % Time Elapsed:  1182.10 s
Iteration:  11 Progress:   7.86  % Time Elapsed:  1276.07 s
Iteration:  12 Progress:   8.57  % Time Elapsed:  1373.08 s
Iteration:  13 Progress:   9.29  % Time Elapsed:  1469.20 s
Iteration:  14 Progress:  10.00  % Time Elapsed:  1533.31 s
Epoch 0 Finished.  Time per Epoch:  1533.31 s
Iteration:  15 Progress:  10.71  % Time Elapsed:  1540.79 s
Iteration:  16 Progress:  11.43  % Time Elapsed:  1547.76 s
Iteration:  17 Progress:  12.14  % Time Elapsed:  1555.12 s
Iteration:  18 Progress:  12.86  % Time Elapsed:  1562.45 s
Iteration:  19 Progress:  13.57  % Time Elapsed:  1569.58 s
Iteration:  20 Progress:  14.29  % Time Elapsed:  1577.24 s
Iteration:  21 Progress:  15.00  % Time Elapsed:  1584.12 s
Iteration:  22 Progress:  15.71  % Time Elapsed:  1591.82 s
Iteration:  23 Progress:  16.43  % Time Elapsed:  1598.68 s
Iteration:  24 Progress:  17.14  % Time Elapsed:  1606.30 s
Iteration:  25 Progress:  17.86  % Time Elapsed:  1613.09 s
Iteration:  26 Progress:  18.57  % Time Elapsed:  1620.83 s
Iteration:  27 Progress:  19.29  % Time Elapsed:  1627.92 s
Iteration:  28 Progress:  20.00  % Time Elapsed:  1634.99 s
Epoch 1 Finished.  Time per Epoch:  817.50 s
Iteration:  29 Progress:  20.71  % Time Elapsed:  1643.35 s
Iteration:  30 Progress:  21.43  % Time Elapsed:  1651.03 s
Iteration:  31 Progress:  22.14  % Time Elapsed:  1659.39 s
Iteration:  32 Progress:  22.86  % Time Elapsed:  1666.70 s
Iteration:  33 Progress:  23.57  % Time Elapsed:  1674.57 s
Iteration:  34 Progress:  24.29  % Time Elapsed:  1681.72 s
Iteration:  35 Progress:  25.00  % Time Elapsed:  1689.09 s
Iteration:  36 Progress:  25.71  % Time Elapsed:  1696.58 s
Iteration:  37 Progress:  26.43  % Time Elapsed:  1703.61 s
Iteration:  38 Progress:  27.14  % Time Elapsed:  1711.38 s
Iteration:  39 Progress:  27.86  % Time Elapsed:  1718.33 s
Iteration:  40 Progress:  28.57  % Time Elapsed:  1726.03 s
Iteration:  41 Progress:  29.29  % Time Elapsed:  1732.85 s
Iteration:  42 Progress:  30.00  % Time Elapsed:  1739.81 s
Epoch 2 Finished.  Time per Epoch:  579.94 s
Iteration:  43 Progress:  30.71  % Time Elapsed:  1748.01 s
Iteration:  44 Progress:  31.43  % Time Elapsed:  1756.53 s
Iteration:  45 Progress:  32.14  % Time Elapsed:  1763.45 s
Iteration:  46 Progress:  32.86  % Time Elapsed:  1770.92 s
Iteration:  47 Progress:  33.57  % Time Elapsed:  1778.17 s
Iteration:  48 Progress:  34.29  % Time Elapsed:  1785.22 s
Iteration:  49 Progress:  35.00  % Time Elapsed:  1792.92 s
Iteration:  50 Progress:  35.71  % Time Elapsed:  1799.65 s
Iteration:  51 Progress:  36.43  % Time Elapsed:  1807.23 s
Iteration:  52 Progress:  37.14  % Time Elapsed:  1814.05 s
Iteration:  53 Progress:  37.86  % Time Elapsed:  1821.71 s
Iteration:  54 Progress:  38.57  % Time Elapsed:  1828.63 s
Iteration:  55 Progress:  39.29  % Time Elapsed:  1836.25 s
Iteration:  56 Progress:  40.00  % Time Elapsed:  1842.48 s
Epoch 3 Finished.  Time per Epoch:  460.62 s
Iteration:  57 Progress:  40.71  % Time Elapsed:  1850.09 s
Iteration:  58 Progress:  41.43  % Time Elapsed:  1857.26 s
```

```
Iteration:  59 Progress:  42.14  % Time Elapsed:  1864.50 s
Iteration:  60 Progress:  42.86  % Time Elapsed:  1872.20 s
Iteration:  61 Progress:  43.57  % Time Elapsed:  1879.09 s
Iteration:  62 Progress:  44.29  % Time Elapsed:  1886.76 s
Iteration:  63 Progress:  45.00  % Time Elapsed:  1893.69 s
Iteration:  64 Progress:  45.71  % Time Elapsed:  1901.40 s
Iteration:  65 Progress:  46.43  % Time Elapsed:  1908.25 s
Iteration:  66 Progress:  47.14  % Time Elapsed:  1915.91 s
Iteration:  67 Progress:  47.86  % Time Elapsed:  1922.98 s
Iteration:  68 Progress:  48.57  % Time Elapsed:  1930.45 s
Iteration:  69 Progress:  49.29  % Time Elapsed:  1937.86 s
Iteration:  70 Progress:  50.00  % Time Elapsed:  1945.33 s
Epoch 4 Finished.  Time per Epoch:  389.07 s
Iteration:  71 Progress:  50.71  % Time Elapsed:  1952.97 s
Iteration:  72 Progress:  51.43  % Time Elapsed:  1959.70 s
Iteration:  73 Progress:  52.14  % Time Elapsed:  1967.44 s
Iteration:  74 Progress:  52.86  % Time Elapsed:  1974.36 s
Iteration:  75 Progress:  53.57  % Time Elapsed:  1983.07 s
Iteration:  76 Progress:  54.29  % Time Elapsed:  1989.95 s
Iteration:  77 Progress:  55.00  % Time Elapsed:  1997.66 s
Iteration:  78 Progress:  55.71  % Time Elapsed:  2004.80 s
Iteration:  79 Progress:  56.43  % Time Elapsed:  2012.14 s
Iteration:  80 Progress:  57.14  % Time Elapsed:  2019.58 s
Iteration:  81 Progress:  57.86  % Time Elapsed:  2026.71 s
Iteration:  82 Progress:  58.57  % Time Elapsed:  2034.25 s
Iteration:  83 Progress:  59.29  % Time Elapsed:  2041.03 s
Iteration:  84 Progress:  60.00  % Time Elapsed:  2048.22 s
Epoch 5 Finished.  Time per Epoch:  341.37 s
Iteration:  85 Progress:  60.71  % Time Elapsed:  2055.10 s
Iteration:  86 Progress:  61.43  % Time Elapsed:  2062.81 s
Iteration:  87 Progress:  62.14  % Time Elapsed:  2069.64 s
Iteration:  88 Progress:  62.86  % Time Elapsed:  2077.19 s
Iteration:  89 Progress:  63.57  % Time Elapsed:  2084.11 s
Iteration:  90 Progress:  64.29  % Time Elapsed:  2091.87 s
Iteration:  91 Progress:  65.00  % Time Elapsed:  2099.50 s
Iteration:  92 Progress:  65.71  % Time Elapsed:  2106.50 s
Iteration:  93 Progress:  66.43  % Time Elapsed:  2114.18 s
Iteration:  94 Progress:  67.14  % Time Elapsed:  2121.01 s
Iteration:  95 Progress:  67.86  % Time Elapsed:  2128.70 s
Iteration:  96 Progress:  68.57  % Time Elapsed:  2135.61 s
Iteration:  97 Progress:  69.29  % Time Elapsed:  2143.28 s
Iteration:  98 Progress:  70.00  % Time Elapsed:  2149.49 s
Epoch 6 Finished.  Time per Epoch:  307.07 s
Iteration:  99 Progress:  70.71  % Time Elapsed:  2158.30 s
Iteration:  100 Progress:  71.43  % Time Elapsed:  2165.10 s
Iteration:  101 Progress:  72.14  % Time Elapsed:  2172.79 s
Iteration:  102 Progress:  72.86  % Time Elapsed:  2179.92 s
Iteration:  103 Progress:  73.57  % Time Elapsed:  2187.13 s
Iteration:  104 Progress:  74.29  % Time Elapsed:  2194.99 s
Iteration:  105 Progress:  75.00  % Time Elapsed:  2202.60 s
Iteration:  106 Progress:  75.71  % Time Elapsed:  2210.48 s
Iteration:  107 Progress:  76.43  % Time Elapsed:  2217.40 s
Iteration:  108 Progress:  77.14  % Time Elapsed:  2225.09 s
Iteration:  109 Progress:  77.86  % Time Elapsed:  2232.19 s
Iteration:  110 Progress:  78.57  % Time Elapsed:  2239.72 s
Iteration:  111 Progress:  79.29  % Time Elapsed:  2247.34 s
Iteration:  112 Progress:  80.00  % Time Elapsed:  2253.89 s
Epoch 7 Finished.  Time per Epoch:  281.74 s
Iteration:  113 Progress:  80.71  % Time Elapsed:  2261.67 s
Iteration:  114 Progress:  81.43  % Time Elapsed:  2268.64 s
Iteration:  115 Progress:  82.14  % Time Elapsed:  2276.43 s
Iteration:  116 Progress:  82.86  % Time Elapsed:  2283.34 s
Iteration:  117 Progress:  83.57  % Time Elapsed:  2291.17 s
Iteration:  118 Progress:  84.29  % Time Elapsed:  2298.35 s
```

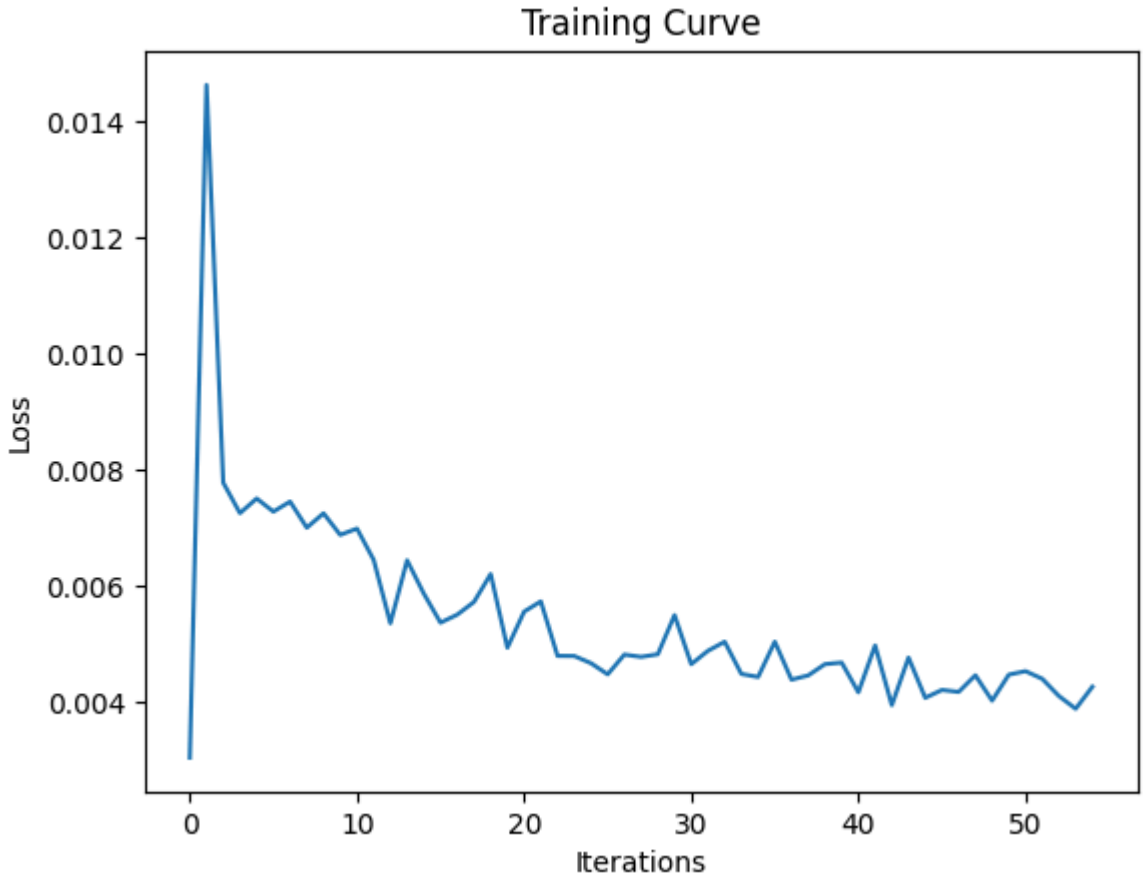
```
Iteration:  119 Progress:  85.00  % Time Elapsed:  2305.99 s
Iteration:  120 Progress:  85.71  % Time Elapsed:  2313.65 s
Iteration:  121 Progress:  86.43  % Time Elapsed:  2320.80 s
Iteration:  122 Progress:  87.14  % Time Elapsed:  2328.50 s
Iteration:  123 Progress:  87.86  % Time Elapsed:  2335.53 s
Iteration:  124 Progress:  88.57  % Time Elapsed:  2343.31 s
Iteration:  125 Progress:  89.29  % Time Elapsed:  2350.21 s
Iteration:  126 Progress:  90.00  % Time Elapsed:  2357.26 s
Epoch 8 Finished.  Time per Epoch:  261.92 s
Iteration:  127 Progress:  90.71  % Time Elapsed:  2365.08 s
Iteration:  128 Progress:  91.43  % Time Elapsed:  2372.52 s
Iteration:  129 Progress:  92.14  % Time Elapsed:  2379.85 s
Iteration:  130 Progress:  92.86  % Time Elapsed:  2387.00 s
Iteration:  131 Progress:  93.57  % Time Elapsed:  2394.60 s
Iteration:  132 Progress:  94.29  % Time Elapsed:  2401.44 s
Iteration:  133 Progress:  95.00  % Time Elapsed:  2409.15 s
Iteration:  134 Progress:  95.71  % Time Elapsed:  2416.13 s
Iteration:  135 Progress:  96.43  % Time Elapsed:  2424.64 s
Iteration:  136 Progress:  97.14  % Time Elapsed:  2431.60 s
Iteration:  137 Progress:  97.86  % Time Elapsed:  2439.05 s
Iteration:  138 Progress:  98.57  % Time Elapsed:  2446.23 s
Iteration:  139 Progress:  99.29  % Time Elapsed:  2453.32 s
Iteration:  140 Progress:  100.00  % Time Elapsed:  2460.17 s
Epoch 9 Finished.  Time per Epoch:  246.02 s
```

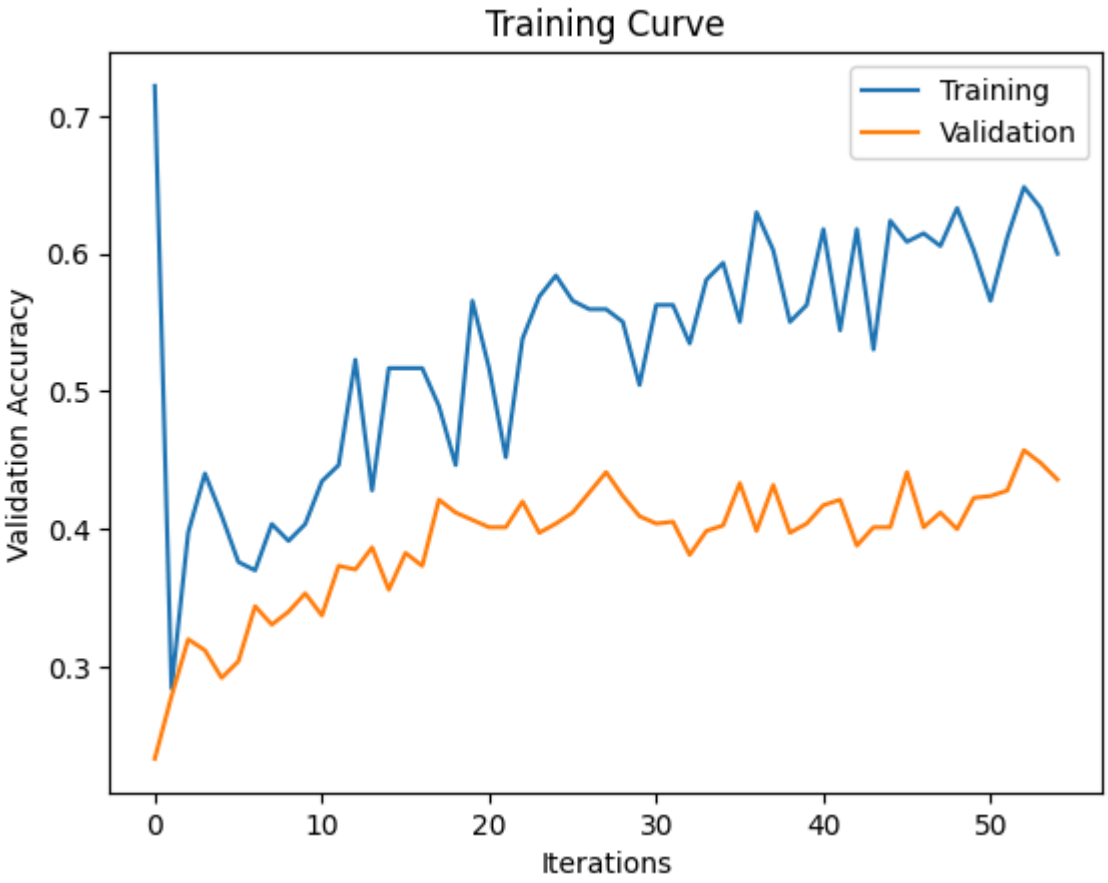



```
Final Training Accuracy: 0.7222857142857143
Final Validation Accuracy: 0.468
Total time:   2460.17 s  Time per Epoch:  246.02 s
```

In [ ]:
```
#Trying Batch size 300, learning rate = 0.003
use_cuda = True
torch.cuda.empty_cache()
model_alex = alexnet

model_name = "alexnet"


if use_cuda and torch.cuda.is_available():
  alexnet.cuda()
  model_alex.cuda()
  print('CUDA is available!  Training on GPU ...')
else:
  print('CUDA is not available.  Training on CPU ...')

# train model
train(model_alex, model_name,learning_rate = 0.003, batch_size=300, num_epochs=5)
```

```
CUDA is not available.  Training on CPU ...
Using alexnet
Iteration:  1 Progress:   1.82  % Time Elapsed:  66.16 s
Iteration:  2 Progress:   3.64  % Time Elapsed:  130.88 s
Iteration:  3 Progress:   5.45  % Time Elapsed:  199.43 s
Iteration:  4 Progress:   7.27  % Time Elapsed:  267.18 s
Iteration:  5 Progress:   9.09  % Time Elapsed:  332.79 s
Iteration:  6 Progress:  10.91  % Time Elapsed:  395.16 s
Iteration:  7 Progress:  12.73  % Time Elapsed:  458.08 s
Iteration:  8 Progress:  14.55  % Time Elapsed:  520.70 s
Iteration:  9 Progress:  16.36  % Time Elapsed:  582.79 s
Iteration:  10 Progress:  18.18  % Time Elapsed:  645.14 s
Iteration:  11 Progress:  20.00  % Time Elapsed:  705.28 s
Epoch 0 Finished.  Time per Epoch:  705.28 s
Iteration:  12 Progress:  21.82  % Time Elapsed:  771.98 s
Iteration:  13 Progress:  23.64  % Time Elapsed:  840.10 s
Iteration:  14 Progress:  25.45  % Time Elapsed:  909.20 s
Iteration:  15 Progress:  27.27  % Time Elapsed:  977.32 s
Iteration:  16 Progress:  29.09  % Time Elapsed:  1044.87 s
Iteration:  17 Progress:  30.91  % Time Elapsed:  1111.87 s
Iteration:  18 Progress:  32.73  % Time Elapsed:  1180.16 s
Iteration:  19 Progress:  34.55  % Time Elapsed:  1247.92 s
Iteration:  20 Progress:  36.36  % Time Elapsed:  1314.17 s
Iteration:  21 Progress:  38.18  % Time Elapsed:  1382.23 s
Iteration:  22 Progress:  40.00  % Time Elapsed:  1441.90 s
Epoch 1 Finished.  Time per Epoch:  720.95 s
Iteration:  23 Progress:  41.82  % Time Elapsed:  1509.15 s
Iteration:  24 Progress:  43.64  % Time Elapsed:  1576.94 s
Iteration:  25 Progress:  45.45  % Time Elapsed:  1643.81 s
Iteration:  26 Progress:  47.27  % Time Elapsed:  1710.59 s
Iteration:  27 Progress:  49.09  % Time Elapsed:  1778.23 s
Iteration:  28 Progress:  50.91  % Time Elapsed:  1845.53 s
Iteration:  29 Progress:  52.73  % Time Elapsed:  1913.83 s
Iteration:  30 Progress:  54.55  % Time Elapsed:  1979.96 s
Iteration:  31 Progress:  56.36  % Time Elapsed:  2047.81 s
Iteration:  32 Progress:  58.18  % Time Elapsed:  2116.59 s
Iteration:  33 Progress:  60.00  % Time Elapsed:  2174.88 s
Epoch 2 Finished.  Time per Epoch:  724.96 s
Iteration:  34 Progress:  61.82  % Time Elapsed:  2242.98 s
Iteration:  35 Progress:  63.64  % Time Elapsed:  2310.78 s
Iteration:  36 Progress:  65.45  % Time Elapsed:  2399.69 s
Iteration:  37 Progress:  67.27  % Time Elapsed:  2469.09 s
Iteration:  38 Progress:  69.09  % Time Elapsed:  2536.71 s
Iteration:  39 Progress:  70.91  % Time Elapsed:  2603.59 s
Iteration:  40 Progress:  72.73  % Time Elapsed:  2671.22 s
Iteration:  41 Progress:  74.55  % Time Elapsed:  2739.65 s
Iteration:  42 Progress:  76.36  % Time Elapsed:  2807.31 s
Iteration:  43 Progress:  78.18  % Time Elapsed:  2874.43 s
Iteration:  44 Progress:  80.00  % Time Elapsed:  2933.60 s
Epoch 3 Finished.  Time per Epoch:  733.40 s
Iteration:  45 Progress:  81.82  % Time Elapsed:  2998.97 s
Iteration:  46 Progress:  83.64  % Time Elapsed:  3061.88 s
Iteration:  47 Progress:  85.45  % Time Elapsed:  3124.55 s
Iteration:  48 Progress:  87.27  % Time Elapsed:  3186.12 s
Iteration:  49 Progress:  89.09  % Time Elapsed:  3248.07 s
Iteration:  50 Progress:  90.91  % Time Elapsed:  3311.33 s
Iteration:  51 Progress:  92.73  % Time Elapsed:  3374.83 s
Iteration:  52 Progress:  94.55  % Time Elapsed:  3437.94 s
Iteration:  53 Progress:  96.36  % Time Elapsed:  3500.27 s
Iteration:  54 Progress:  98.18  % Time Elapsed:  3562.73 s
Iteration:  55 Progress:  100.00  % Time Elapsed:  3618.25 s
Epoch 4 Finished.  Time per Epoch:  723.65 s
```



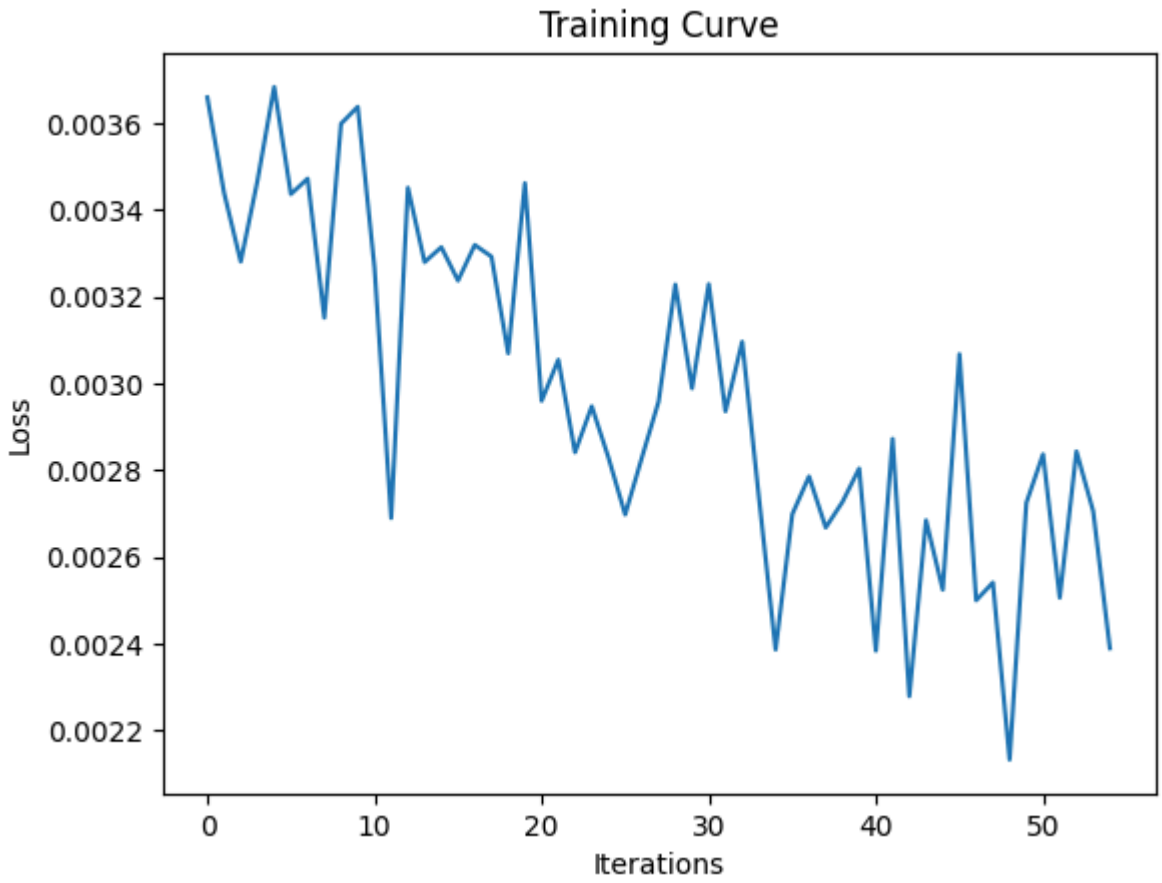


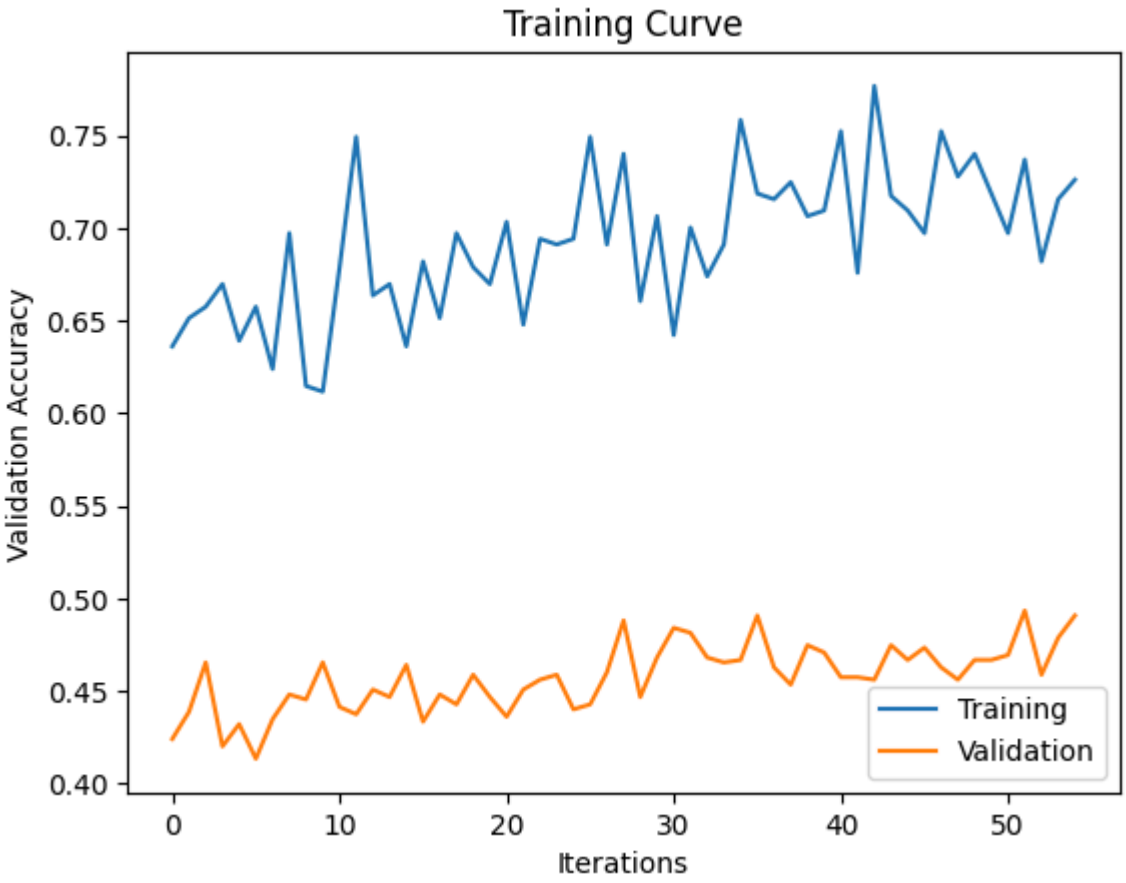

```
Final Training Accuracy: 0.6265714285714286
Final Validation Accuracy: 0.436
Total time:   3618.25 s  Time per Epoch:  723.65 s
```

In [ ]:
```
# now tring batch size = 327, learning rate = 0.0008
use_cuda = True
torch.cuda.empty_cache()
model_alex = alexnet
```

```
model_name = "alexnet"


if use_cuda and torch.cuda.is_available():
  alexnet.cuda()
  model_alex.cuda()
  print('CUDA is available!  Training on GPU ...')
else:
  print('CUDA is not available.  Training on CPU ...')

# train model
train(model_alex, model_name,learning_rate = 0.0008, batch_size=327, num_epochs=5)
```

```
CUDA is not available.  Training on CPU ...
Using alexnet
Iteration:  1 Progress:   1.82  % Time Elapsed:  63.76 s
Iteration:  2 Progress:   3.64  % Time Elapsed:  126.48 s
Iteration:  3 Progress:   5.45  % Time Elapsed:  188.40 s
Iteration:  4 Progress:   7.27  % Time Elapsed:  252.41 s
Iteration:  5 Progress:   9.09  % Time Elapsed:  315.86 s
Iteration:  6 Progress:  10.91  % Time Elapsed:  378.95 s
Iteration:  7 Progress:  12.73  % Time Elapsed:  441.19 s
Iteration:  8 Progress:  14.55  % Time Elapsed:  505.08 s
Iteration:  9 Progress:  16.36  % Time Elapsed:  574.25 s
Iteration:  10 Progress:  18.18  % Time Elapsed:  637.35 s
Iteration:  11 Progress:  20.00  % Time Elapsed:  690.81 s
Epoch 0 Finished.  Time per Epoch:  690.81 s
Iteration:  12 Progress:  21.82  % Time Elapsed:  752.82 s
Iteration:  13 Progress:  23.64  % Time Elapsed:  816.38 s
Iteration:  14 Progress:  25.45  % Time Elapsed:  879.93 s
Iteration:  15 Progress:  27.27  % Time Elapsed:  942.99 s
Iteration:  16 Progress:  29.09  % Time Elapsed:  1005.74 s
Iteration:  17 Progress:  30.91  % Time Elapsed:  1068.09 s
Iteration:  18 Progress:  32.73  % Time Elapsed:  1130.74 s
Iteration:  19 Progress:  34.55  % Time Elapsed:  1193.18 s
Iteration:  20 Progress:  36.36  % Time Elapsed:  1254.45 s
Iteration:  21 Progress:  38.18  % Time Elapsed:  1316.97 s
Iteration:  22 Progress:  40.00  % Time Elapsed:  1372.01 s
Epoch 1 Finished.  Time per Epoch:  686.01 s
Iteration:  23 Progress:  41.82  % Time Elapsed:  1438.72 s
Iteration:  24 Progress:  43.64  % Time Elapsed:  1506.65 s
Iteration:  25 Progress:  45.45  % Time Elapsed:  1574.50 s
Iteration:  26 Progress:  47.27  % Time Elapsed:  1640.81 s
Iteration:  27 Progress:  49.09  % Time Elapsed:  1708.43 s
Iteration:  28 Progress:  50.91  % Time Elapsed:  1777.00 s
Iteration:  29 Progress:  52.73  % Time Elapsed:  1843.55 s
Iteration:  30 Progress:  54.55  % Time Elapsed:  1910.92 s
Iteration:  31 Progress:  56.36  % Time Elapsed:  1979.52 s
Iteration:  32 Progress:  58.18  % Time Elapsed:  2046.93 s
Iteration:  33 Progress:  60.00  % Time Elapsed:  2104.74 s
Epoch 2 Finished.  Time per Epoch:  701.58 s
Iteration:  34 Progress:  61.82  % Time Elapsed:  2172.52 s
Iteration:  35 Progress:  63.64  % Time Elapsed:  2240.22 s
Iteration:  36 Progress:  65.45  % Time Elapsed:  2306.45 s
Iteration:  37 Progress:  67.27  % Time Elapsed:  2373.97 s
Iteration:  38 Progress:  69.09  % Time Elapsed:  2441.81 s
Iteration:  39 Progress:  70.91  % Time Elapsed:  2507.78 s
Iteration:  40 Progress:  72.73  % Time Elapsed:  2575.43 s
Iteration:  41 Progress:  74.55  % Time Elapsed:  2643.22 s
Iteration:  42 Progress:  76.36  % Time Elapsed:  2709.70 s
Iteration:  43 Progress:  78.18  % Time Elapsed:  2777.07 s
Iteration:  44 Progress:  80.00  % Time Elapsed:  2836.75 s
Epoch 3 Finished.  Time per Epoch:  709.19 s
Iteration:  45 Progress:  81.82  % Time Elapsed:  2904.26 s
Iteration:  46 Progress:  83.64  % Time Elapsed:  2971.47 s
Iteration:  47 Progress:  85.45  % Time Elapsed:  3039.16 s
Iteration:  48 Progress:  87.27  % Time Elapsed:  3106.11 s
Iteration:  49 Progress:  89.09  % Time Elapsed:  3173.41 s
Iteration:  50 Progress:  90.91  % Time Elapsed:  3240.98 s
Iteration:  51 Progress:  92.73  % Time Elapsed:  3307.71 s
Iteration:  52 Progress:  94.55  % Time Elapsed:  3375.10 s
Iteration:  53 Progress:  96.36  % Time Elapsed:  3442.81 s
Iteration:  54 Progress:  98.18  % Time Elapsed:  3509.87 s
Iteration:  55 Progress:  100.00  % Time Elapsed:  3569.05 s
Epoch 4 Finished.  Time per Epoch:  713.81 s
```



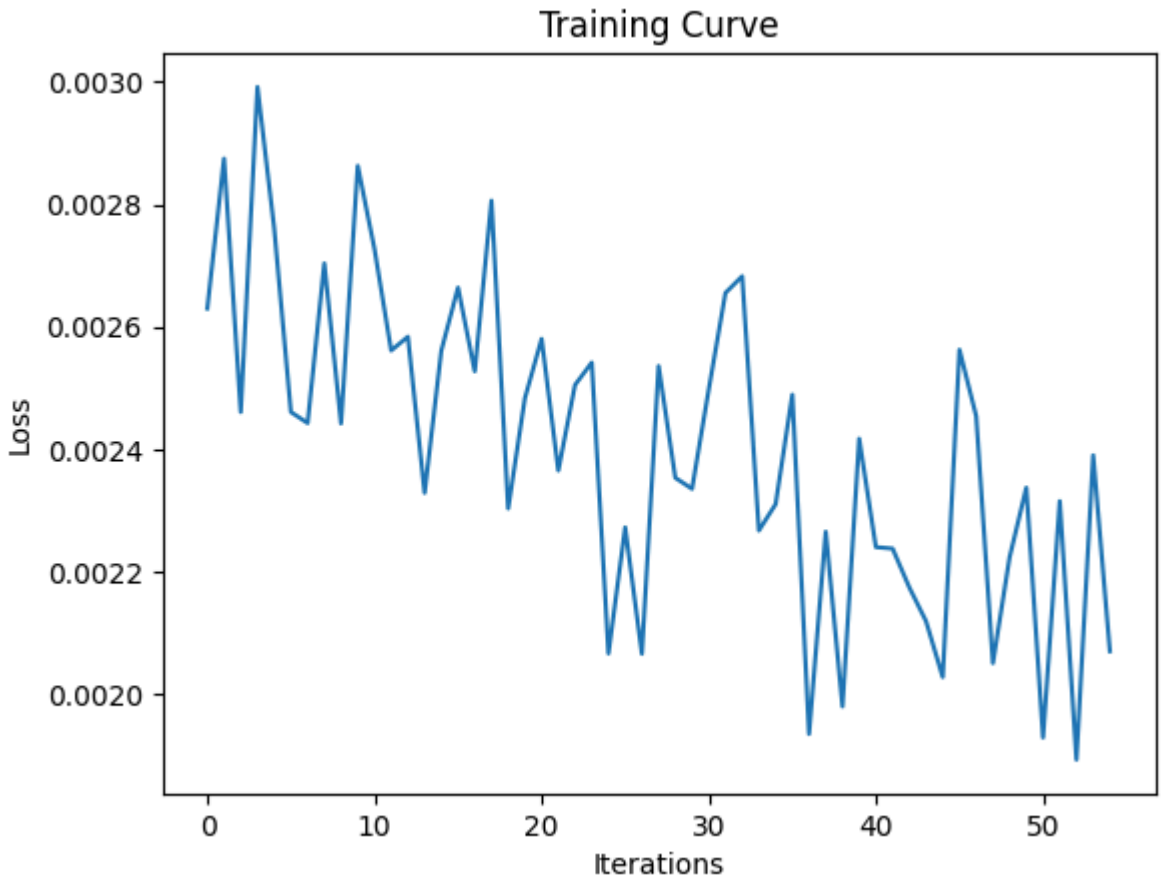


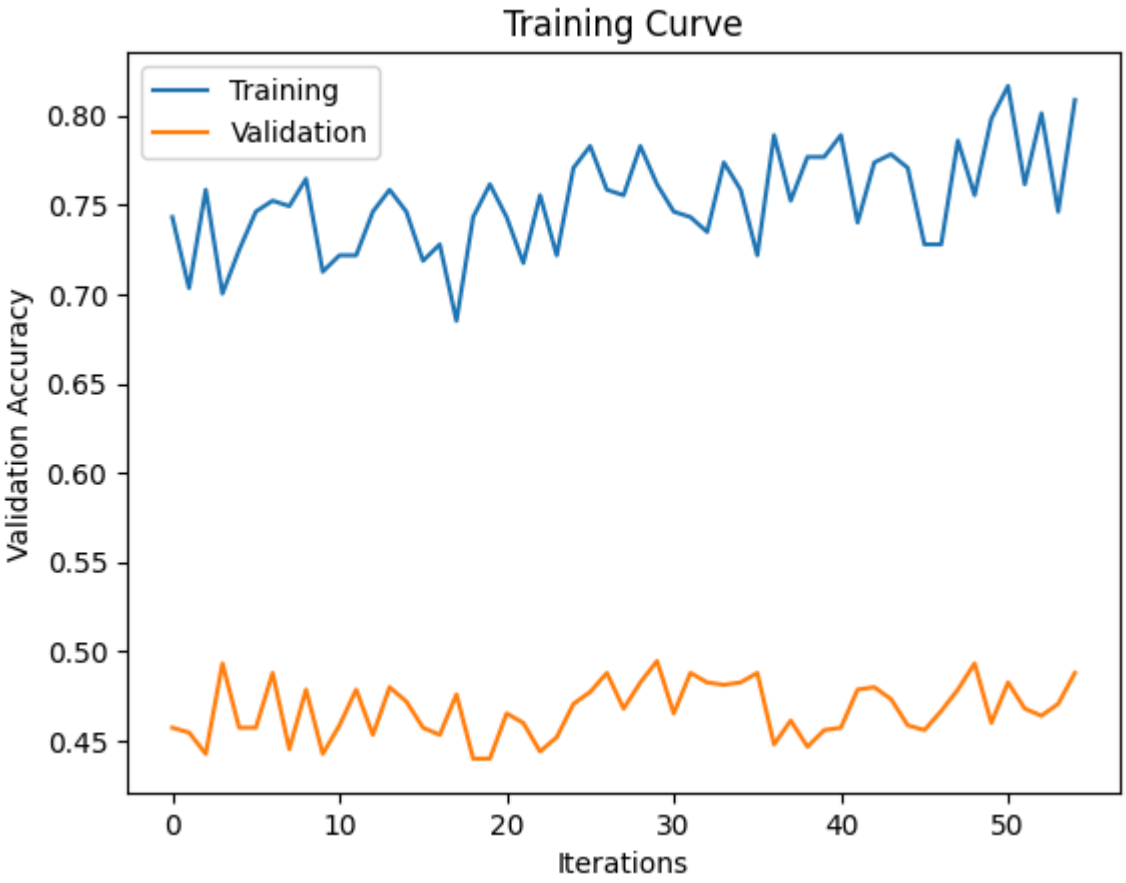

```
Final Training Accuracy: 0.7377142857142858
Final Validation Accuracy: 0.4906666666666664
Total time:   3569.06 s  Time per Epoch:  713.81 s
```

In [ ]:
```
# now tring batch size = 327, learning rate = 0.0015
use_cuda = True
torch.cuda.empty_cache()
model_alex = alexnet
```

```
model_name = "alexnet"


if use_cuda and torch.cuda.is_available():
  alexnet.cuda()
  model_alex.cuda()
  print('CUDA is available!  Training on GPU ...')
else:
  print('CUDA is not available.  Training on CPU ...')

# train model
train(model_alex, model_name,learning_rate = 0.00015, batch_size=327, num_epochs=5)
```

```
CUDA is not available.  Training on CPU ...
Using alexnet
Iteration:  1 Progress:   1.82  % Time Elapsed:  68.05 s
Iteration:  2 Progress:   3.64  % Time Elapsed:  136.60 s
Iteration:  3 Progress:   5.45  % Time Elapsed:  202.75 s
Iteration:  4 Progress:   7.27  % Time Elapsed:  270.14 s
Iteration:  5 Progress:   9.09  % Time Elapsed:  339.02 s
Iteration:  6 Progress:  10.91  % Time Elapsed:  406.66 s
Iteration:  7 Progress:  12.73  % Time Elapsed:  473.76 s
Iteration:  8 Progress:  14.55  % Time Elapsed:  541.07 s
Iteration:  9 Progress:  16.36  % Time Elapsed:  609.13 s
Iteration:  10 Progress:  18.18  % Time Elapsed:  676.33 s
Iteration:  11 Progress:  20.00  % Time Elapsed:  735.47 s
Epoch 0 Finished.  Time per Epoch:  735.48 s
Iteration:  12 Progress:  21.82  % Time Elapsed:  804.00 s
Iteration:  13 Progress:  23.64  % Time Elapsed:  871.27 s
Iteration:  14 Progress:  25.45  % Time Elapsed:  938.51 s
Iteration:  15 Progress:  27.27  % Time Elapsed:  1006.37 s
Iteration:  16 Progress:  29.09  % Time Elapsed:  1074.63 s
Iteration:  17 Progress:  30.91  % Time Elapsed:  1144.81 s
Iteration:  18 Progress:  32.73  % Time Elapsed:  1211.43 s
Iteration:  19 Progress:  34.55  % Time Elapsed:  1279.37 s
Iteration:  20 Progress:  36.36  % Time Elapsed:  1347.60 s
Iteration:  21 Progress:  38.18  % Time Elapsed:  1415.07 s
Iteration:  22 Progress:  40.00  % Time Elapsed:  1472.92 s
Epoch 1 Finished.  Time per Epoch:  736.46 s
Iteration:  23 Progress:  41.82  % Time Elapsed:  1541.21 s
Iteration:  24 Progress:  43.64  % Time Elapsed:  1609.41 s
Iteration:  25 Progress:  45.45  % Time Elapsed:  1675.66 s
Iteration:  26 Progress:  47.27  % Time Elapsed:  1743.96 s
Iteration:  27 Progress:  49.09  % Time Elapsed:  1812.02 s
Iteration:  28 Progress:  50.91  % Time Elapsed:  1879.63 s
Iteration:  29 Progress:  52.73  % Time Elapsed:  1946.96 s
Iteration:  30 Progress:  54.55  % Time Elapsed:  2015.04 s
Iteration:  31 Progress:  56.36  % Time Elapsed:  2082.82 s
Iteration:  32 Progress:  58.18  % Time Elapsed:  2150.93 s
Iteration:  33 Progress:  60.00  % Time Elapsed:  2209.55 s
Epoch 2 Finished.  Time per Epoch:  736.52 s
Iteration:  34 Progress:  61.82  % Time Elapsed:  2277.40 s
Iteration:  35 Progress:  63.64  % Time Elapsed:  2344.07 s
Iteration:  36 Progress:  65.45  % Time Elapsed:  2416.69 s
Iteration:  37 Progress:  67.27  % Time Elapsed:  2488.55 s
Iteration:  38 Progress:  69.09  % Time Elapsed:  2558.08 s
Iteration:  39 Progress:  70.91  % Time Elapsed:  2624.31 s
Iteration:  40 Progress:  72.73  % Time Elapsed:  2693.04 s
Iteration:  41 Progress:  74.55  % Time Elapsed:  2761.66 s
Iteration:  42 Progress:  76.36  % Time Elapsed:  2829.82 s
Iteration:  43 Progress:  78.18  % Time Elapsed:  2897.58 s
Iteration:  44 Progress:  80.00  % Time Elapsed:  2956.79 s
Epoch 3 Finished.  Time per Epoch:  739.20 s
Iteration:  45 Progress:  81.82  % Time Elapsed:  3025.02 s
Iteration:  46 Progress:  83.64  % Time Elapsed:  3092.49 s
Iteration:  47 Progress:  85.45  % Time Elapsed:  3160.68 s
Iteration:  48 Progress:  87.27  % Time Elapsed:  3228.79 s
Iteration:  49 Progress:  89.09  % Time Elapsed:  3299.28 s
Iteration:  50 Progress:  90.91  % Time Elapsed:  3367.83 s
Iteration:  51 Progress:  92.73  % Time Elapsed:  3434.68 s
Iteration:  52 Progress:  94.55  % Time Elapsed:  3502.10 s
Iteration:  53 Progress:  96.36  % Time Elapsed:  3570.66 s
Iteration:  54 Progress:  98.18  % Time Elapsed:  3638.28 s
Iteration:  55 Progress:  100.00  % Time Elapsed:  3696.44 s
Epoch 4 Finished.  Time per Epoch:  739.29 s
```



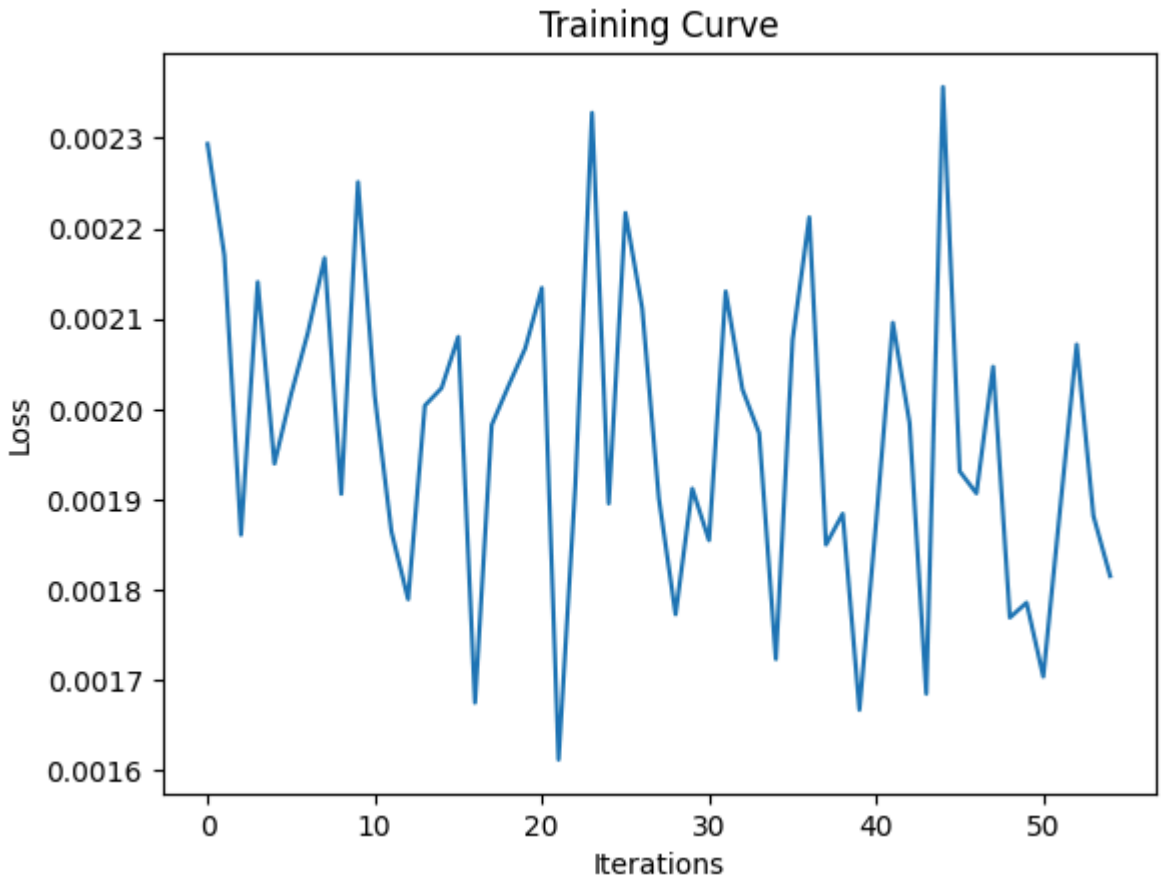


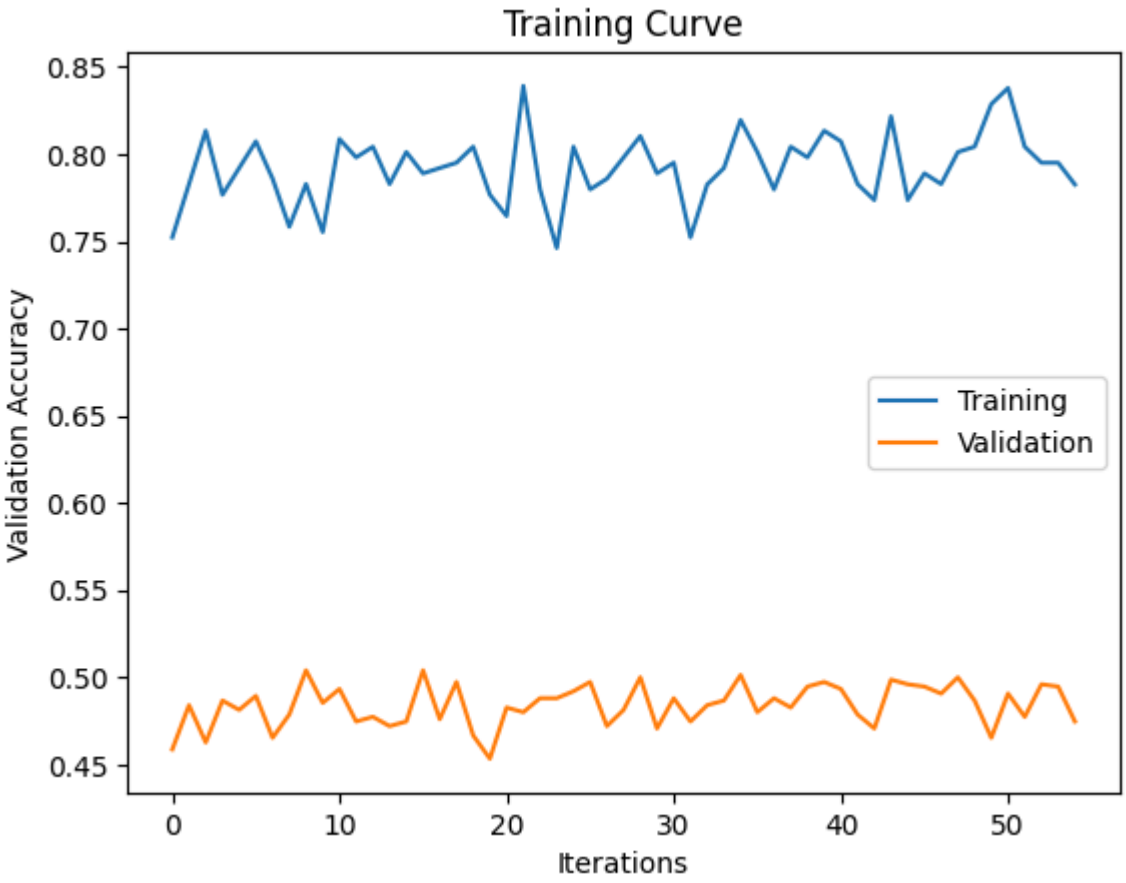

```
Final Training Accuracy: 0.7742857142857142
Final Validation Accuracy: 0.488
Total time:   3696.44 s  Time per Epoch:  739.29 s
```

In [ ]:
```
# now tring batch size = 327, learning rate = 0.0018
use_cuda = True
torch.cuda.empty_cache()
model_alex = alexnet
```

```
model_name = "alexnet"


if use_cuda and torch.cuda.is_available():
  alexnet.cuda()
  model_alex.cuda()
  print('CUDA is available!  Training on GPU ...')
else:
  print('CUDA is not available.  Training on CPU ...')

# train model
train(model_alex, model_name,learning_rate = 0.00018, batch_size=327, num_epochs=5)
```

```
CUDA is not available.  Training on CPU ...
Using alexnet
Iteration:  1 Progress:   1.82  % Time Elapsed:  67.20 s
Iteration:  2 Progress:   3.64  % Time Elapsed:  135.89 s
Iteration:  3 Progress:   5.45  % Time Elapsed:  205.13 s
Iteration:  4 Progress:   7.27  % Time Elapsed:  274.43 s
Iteration:  5 Progress:   9.09  % Time Elapsed:  345.52 s
Iteration:  6 Progress:  10.91  % Time Elapsed:  414.79 s
Iteration:  7 Progress:  12.73  % Time Elapsed:  484.49 s
Iteration:  8 Progress:  14.55  % Time Elapsed:  554.13 s
Iteration:  9 Progress:  16.36  % Time Elapsed:  623.60 s
Iteration:  10 Progress:  18.18  % Time Elapsed:  691.84 s
Iteration:  11 Progress:  20.00  % Time Elapsed:  750.82 s
Epoch 0 Finished.  Time per Epoch:  750.83 s
Iteration:  12 Progress:  21.82  % Time Elapsed:  820.70 s
Iteration:  13 Progress:  23.64  % Time Elapsed:  889.77 s
Iteration:  14 Progress:  25.45  % Time Elapsed:  958.97 s
Iteration:  15 Progress:  27.27  % Time Elapsed:  1028.22 s
Iteration:  16 Progress:  29.09  % Time Elapsed:  1097.17 s
Iteration:  17 Progress:  30.91  % Time Elapsed:  1165.77 s
Iteration:  18 Progress:  32.73  % Time Elapsed:  1233.11 s
Iteration:  19 Progress:  34.55  % Time Elapsed:  1301.64 s
Iteration:  20 Progress:  36.36  % Time Elapsed:  1370.40 s
Iteration:  21 Progress:  38.18  % Time Elapsed:  1439.42 s
Iteration:  22 Progress:  40.00  % Time Elapsed:  1500.01 s
Epoch 1 Finished.  Time per Epoch:  750.01 s
Iteration:  23 Progress:  41.82  % Time Elapsed:  1568.86 s
Iteration:  24 Progress:  43.64  % Time Elapsed:  1636.34 s
Iteration:  25 Progress:  45.45  % Time Elapsed:  1705.24 s
Iteration:  26 Progress:  47.27  % Time Elapsed:  1774.06 s
Iteration:  27 Progress:  49.09  % Time Elapsed:  1843.14 s
Iteration:  28 Progress:  50.91  % Time Elapsed:  1912.11 s
Iteration:  29 Progress:  52.73  % Time Elapsed:  1981.30 s
Iteration:  30 Progress:  54.55  % Time Elapsed:  2050.48 s
Iteration:  31 Progress:  56.36  % Time Elapsed:  2118.30 s
Iteration:  32 Progress:  58.18  % Time Elapsed:  2186.23 s
Iteration:  33 Progress:  60.00  % Time Elapsed:  2246.91 s
Epoch 2 Finished.  Time per Epoch:  748.97 s
Iteration:  34 Progress:  61.82  % Time Elapsed:  2316.40 s
Iteration:  35 Progress:  63.64  % Time Elapsed:  2384.96 s
Iteration:  36 Progress:  65.45  % Time Elapsed:  2453.76 s
Iteration:  37 Progress:  67.27  % Time Elapsed:  2520.93 s
Iteration:  38 Progress:  69.09  % Time Elapsed:  2589.54 s
Iteration:  39 Progress:  70.91  % Time Elapsed:  2657.92 s
Iteration:  40 Progress:  72.73  % Time Elapsed:  2726.98 s
Iteration:  41 Progress:  74.55  % Time Elapsed:  2796.07 s
Iteration:  42 Progress:  76.36  % Time Elapsed:  2864.08 s
Iteration:  43 Progress:  78.18  % Time Elapsed:  2932.41 s
Iteration:  44 Progress:  80.00  % Time Elapsed:  2992.75 s
Epoch 3 Finished.  Time per Epoch:  748.19 s
Iteration:  45 Progress:  81.82  % Time Elapsed:  3062.27 s
Iteration:  46 Progress:  83.64  % Time Elapsed:  3131.02 s
Iteration:  47 Progress:  85.45  % Time Elapsed:  3199.88 s
Iteration:  48 Progress:  87.27  % Time Elapsed:  3266.79 s
Iteration:  49 Progress:  89.09  % Time Elapsed:  3334.98 s
Iteration:  50 Progress:  90.91  % Time Elapsed:  3403.75 s
Iteration:  51 Progress:  92.73  % Time Elapsed:  3472.12 s
Iteration:  52 Progress:  94.55  % Time Elapsed:  3540.48 s
Iteration:  53 Progress:  96.36  % Time Elapsed:  3606.97 s
Iteration:  54 Progress:  98.18  % Time Elapsed:  3675.31 s
Iteration:  55 Progress:  100.00  % Time Elapsed:  3734.76 s
Epoch 4 Finished.  Time per Epoch:  746.95 s
```






```
Final Training Accuracy: 0.8011428571428572
Final Validation Accuracy: 0.4746666666666667
Total time:   3734.76 s  Time per Epoch:  746.95 s
```

- Now that we find the hyperparameters, we want to use the test data to see how well our model predicts on unseen data.

In [ ]:
```
test_accuracy = get_accuracy_of_transfer(model_alex, test_loader)
print("Test Accuracy: {:.2f}%".format(test_accuracy * 100))
```

```
Test Accuracy: 78.77%
```

- To gain more testing expernence, we introduce a dog picture of our own to see how our model predicts.

In [ ]:
```
from PIL import Image
image1 = Image.open('/content/gdrive/MyDrive/test_images/dog_with_person.png')
image2 = Image.open('/content/gdrive/MyDrive/test_images/dog_with_elevator.jpg')
image3 = Image.open('/content/gdrive/MyDrive/test_images/dog_with_tree.jpg')
image4 = Image.open('/content/gdrive/MyDrive/test_images/dog1_1.jpg')
image5 = Image.open('/content/gdrive/MyDrive/test_images/dog1_2.jpg')
image6 = Image.open('/content/gdrive/MyDrive/test_images/dog1_3.jpg')
```

In [ ]:
```
# Define data transformations
data_transform = transforms.Compose([
    transforms.Resize((224, 224)),  # Resize to match CNN input
    transforms.ToTensor(),
    transforms.Lambda(lambda image: image[:3, :, :]),  # Select only the first 3 ch
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])  # Normalize us
])
```

In [ ]:
```
image1_tensor = data_transform(image1)
image1_tensor = image1_tensor.unsqueeze(0)

with torch.no_grad():
    output = model_alex(image1_tensor)
    _, predicted = torch.max(output, 1)

print("Predicted class:", predicted.item())
```

```
Predicted class: 21
```

In [ ]:
```
image2_tensor = data_transform(image2)
image2_tensor = image2_tensor.unsqueeze(0)

with torch.no_grad():
    output = model_alex(image2_tensor)
    _, predicted = torch.max(output, 1)

print("Predicted class:", predicted.item())
```

```
Predicted class: 47
```

In [ ]:
```
image3_tensor = data_transform(image2)
image3_tensor = image3_tensor.unsqueeze(0)

with torch.no_grad():
    output = model_alex(image3_tensor)
    _, predicted = torch.max(output, 1)

print("Predicted class:", predicted.item())
```

```
Predicted class: 47
```

In [ ]:
```
image4_tensor = data_transform(image2)
image4_tensor = image4_tensor.unsqueeze(0)

with torch.no_grad():
    output = model_alex(image4_tensor)
```

```
    _, predicted = torch.max(output, 1)

print("Predicted class:", predicted.item())
```

```
Predicted class: 47
```

In [ ]:
```
image5_tensor = data_transform(image2)
image5_tensor = image5_tensor.unsqueeze(0)

with torch.no_grad():
    output = model_alex(image5_tensor)
    _, predicted = torch.max(output, 1)

print("Predicted class:", predicted.item())
```

```
Predicted class: 47
```

In [ ]:
```
image6_tensor = data_transform(image2)
image6_tensor = image6_tensor.unsqueeze(0)

with torch.no_grad():
    output = model_alex(image6_tensor)
    _, predicted = torch.max(output, 1)

print("Predicted class:", predicted.item())
```

```
Predicted class: 47
```

- We see that, for image one our model predicted as n02104029-kuvasz, and for all other visions of the same dog our model predicted as miniature_poodle, which is correct. This may be that image1 is the only image where dog and person are in the same frame, and our model is having difficulties identify with the noise of a person.

In [ ]:
```
selected_classes[47]
```

Out[ ]:
```
'n02113712-miniature_poodle'
```

In [ ]:
```
selected_classes[21]
```

Out[ ]:
```
'n02104029-kuvasz'
```

In [ ]:
```
# Now, we access more images to test out our model. The images are already in test,
```

In [ ]:
```
from PIL import Image
image2_1 = Image.open('/content/gdrive/MyDrive/test_images/dog2_1.jpg')
image2_2 = Image.open('/content/gdrive/MyDrive/test_images/dog2_2.jpg')
image2_3 = Image.open('/content/gdrive/MyDrive/test_images/dog2_3.jpg')
image3_1 = Image.open('/content/gdrive/MyDrive/test_images/dog3_1.jpg')
image3_2 = Image.open('/content/gdrive/MyDrive/test_images/dog3_2.jpg')
image3_3 = Image.open('/content/gdrive/MyDrive/test_images/dog3_3.jpg')
image4_1 = Image.open('/content/gdrive/MyDrive/test_images/dog4_1.jpg')
image4_2 = Image.open('/content/gdrive/MyDrive/test_images/dog4_2.jpg')
image4_3 = Image.open('/content/gdrive/MyDrive/test_images/dog4_3.jpg')
```

In [ ]:
```
```

In [ ]:
```
image2_1_tensor = data_transform(image2_1)
image2_1_tensor = image2_1_tensor.unsqueeze(0)

with torch.no_grad():
```

```
    output = model_alex(image2_1_tensor)
    _, predicted = torch.max(output, 1)

print("Predicted class:", predicted.item())
```

```
Predicted class:11
```

In [ ]:
```
image2_2_tensor = data_transform(image2_2)
image2_2_tensor = image2_2_tensor.unsqueeze(0)

with torch.no_grad():
    output = model_alex(image2_2_tensor)
    _, predicted = torch.max(output, 1)

print("Predicted class:", predicted.item())
```

```
Predicted class:11
```

In [ ]:
```
image2_3_tensor = data_transform(image2_3)
image2_3_tensor = image2_3_tensor.unsqueeze(0)

with torch.no_grad():
    output = model_alex(image2_3_tensor)
    _, predicted = torch.max(output, 1)

print("Predicted class:", predicted.item())
```

```
Predicted class:46
```

In [ ]:
```

```

In [ ]:
```
image3_1_tensor = data_transform(image3_1)
image3_1_tensor = image3_1_tensor.unsqueeze(0)

with torch.no_grad():
    output = model_alex(image3_1_tensor)
    _, predicted = torch.max(output, 1)

print("Predicted class:", predicted.item())
```

```
Predicted class:32
```

In [ ]:
```
image3_2_tensor = data_transform(image3_2)
image3_2_tensor = image3_2_tensor.unsqueeze(0)

with torch.no_grad():
    output = model_alex(image3_2_tensor)
    _, predicted = torch.max(output, 1)

print("Predicted class:", predicted.item())
```

```
Predicted class:32
```

In [ ]:
```
image3_3_tensor = data_transform(image3_3)
image3_3_tensor = image3_3_tensor.unsqueeze(0)

with torch.no_grad():
    output = model_alex(image3_3_tensor)
    _, predicted = torch.max(output, 1)

print("Predicted class:", predicted.item())
```

```
Predicted class:32
```

In [ ]:
```

```

In [ ]:
```
```

In [ ]:
```
image4_1_tensor = data_transform(image4_1)
image4_1_tensor = image4_1_tensor.unsqueeze(0)

with torch.no_grad():
    output = model_alex(image4_1_tensor)
    _, predicted = torch.max(output, 1)

print("Predicted class:", predicted.item())
```

```
Predicted class:18
```

In [ ]:
```
image4_2_tensor = data_transform(image4_2)
image4_2_tensor = image4_2_tensor.unsqueeze(0)

with torch.no_grad():
    output = model_alex(image4_2_tensor)
    _, predicted = torch.max(output, 1)

print("Predicted class:", predicted.item())
```

```
Predicted class:27
```

In [ ]:
```
image4_3_tensor = data_transform(image4_3)
image4_3_tensor = image4_3_tensor.unsqueeze(0)

with torch.no_grad():
    output = model_alex(image4_3_tensor)
    _, predicted = torch.max(output, 1)

print("Predicted class:", predicted.item())
```

```
Predicted class:27
```

In [ ]:
```
# Now, we know that dog 2 is Gordon_setter, dog 3 is Chihuahua, dog 4 is Mexican_ha
# Let us check the resuls
```

In [ ]:
```
selected_classes[11]
```

```
'n02101006-Gordon_setter'
```

In [ ]:
```
selected_classes[46]
```

```
'n02113186-Cardigan'
```

In [ ]:
```
selected_classes[32]
```

```
'n02085620-Chihuahua'
```

In [ ]:
```
selected_classes[18]
```

```
'n02107142-Doberman'
```

In [ ]:
```
selected_classes[27]
```

```
'n02113978-Mexican_hairless'
```

In [ ]:
```
# Now we see that 7 out of the 9 predictions are actually correct. This is 78% accu
```

- Now, in order to further improve our model to detect dogs first (for example sepreate out from person when in the same frame), as this would probably make our model

more accurate, we introduce the YOLO network.

- We will build a two step pipeline, where YOLO is used first and then passed onto Alexnet to complete the whole identification process.

In [ ]:
```
# From later on this point of the document we will reconsider all necessary imports
# We do not want to run over all the code again every time we reopen the session. T
# be ver time consuming to re-run all the trainings.
################################################################################

# https://docs.ultralytics.com/yolov5/tutorials/pytorch_hub_model_loading/#simple-e
# By refering to this document we are able to access the pre trained SSD model to a

################################################################################
# We start Here
```

In [ ]:
```
# Importing Everyhing We Need
from google.colab import drive
from skimage.transform import resize
from sklearn.model_selection import train_test_split
from torch.utils.data import DataLoader
from torch.utils.data.sampler import SubsetRandomSampler
from torchvision import datasets, models, transforms
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
import shutil
import time
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import random
import PIL
# !pip install ultralytics
# from ultralytics import YOLO
```

In [ ]:
```
pip install -r https://raw.githubusercontent.com/ultralytics/yolov5/master/requirem
```

```
Requirement already satisfied: gitpython>=3.1.30 in /usr/local/lib/python3.10/dist
-packages (from -r https://raw.githubusercontent.com/ultralytics/yolov5/master/req
uirements.txt (line 5)) (3.1.43)
Requirement already satisfied: matplotlib>=3.3 in /usr/local/lib/python3.10/dist-p
ackages (from -r https://raw.githubusercontent.com/ultralytics/yolov5/master/requi
rements.txt (line 6)) (3.7.1)
Requirement already satisfied: numpy>=1.23.5 in /usr/local/lib/python3.10/dist-pac
kages (from -r https://raw.githubusercontent.com/ultralytics/yolov5/master/require
ments.txt (line 7)) (1.26.4)
Requirement already satisfied: opencv-python>=4.1.1 in /usr/local/lib/python3.10/d
ist-packages (from -r https://raw.githubusercontent.com/ultralytics/yolov5/master/
requirements.txt (line 8)) (4.10.0.84)
Requirement already satisfied: pillow>=10.3.0 in /usr/local/lib/python3.10/dist-pa
ckages (from -r https://raw.githubusercontent.com/ultralytics/yolov5/master/requir
ements.txt (line 9)) (10.4.0)
Requirement already satisfied: psutil in /usr/local/lib/python3.10/dist-packages
(from -r https://raw.githubusercontent.com/ultralytics/yolov5/master/requirements.
txt (line 10)) (5.9.5)
Requirement already satisfied: PyYAML>=5.3.1 in /usr/local/lib/python3.10/dist-pac
kages (from -r https://raw.githubusercontent.com/ultralytics/yolov5/master/require
ments.txt (line 11)) (6.0.2)
Requirement already satisfied: requests>=2.32.0 in /usr/local/lib/python3.10/dist-
packages (from -r https://raw.githubusercontent.com/ultralytics/yolov5/master/requ
irements.txt (line 12)) (2.32.3)
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.10/dist-pack
ages (from -r https://raw.githubusercontent.com/ultralytics/yolov5/master/requirem
ents.txt (line 13)) (1.13.1)
Collecting thop>=0.1.1 (from -r https://raw.githubusercontent.com/ultralytics/yolo
v5/master/requirements.txt (line 14))
  Downloading thop-0.1.1.post2209072238-py3-none-any.whl.metadata (2.7 kB)
Requirement already satisfied: torch>=1.8.0 in /usr/local/lib/python3.10/dist-pack
ages (from -r https://raw.githubusercontent.com/ultralytics/yolov5/master/requirem
ents.txt (line 15)) (2.3.1+cu121)
Requirement already satisfied: torchvision>=0.9.0 in /usr/local/lib/python3.10/dis
t-packages (from -r https://raw.githubusercontent.com/ultralytics/yolov5/master/re
quirements.txt (line 16)) (0.18.1+cu121)
Requirement already satisfied: tqdm>=4.64.0 in /usr/local/lib/python3.10/dist-pack
ages (from -r https://raw.githubusercontent.com/ultralytics/yolov5/master/requirem
ents.txt (line 17)) (4.66.5)
Requirement already satisfied: ultralytics>=8.2.34 in /usr/local/lib/python3.10/di
st-packages (from -r https://raw.githubusercontent.com/ultralytics/yolov5/master/r
equirements.txt (line 18)) (8.2.76)
Requirement already satisfied: pandas>=1.1.4 in /usr/local/lib/python3.10/dist-pac
kages (from -r https://raw.githubusercontent.com/ultralytics/yolov5/master/require
ments.txt (line 27)) (2.1.4)
Requirement already satisfied: seaborn>=0.11.0 in /usr/local/lib/python3.10/dist-p
ackages (from -r https://raw.githubusercontent.com/ultralytics/yolov5/master/requi
rements.txt (line 28)) (0.13.1)
Requirement already satisfied: setuptools>=70.0.0 in /usr/local/lib/python3.10/dis
t-packages (from -r https://raw.githubusercontent.com/ultralytics/yolov5/master/re
quirements.txt (line 42)) (71.0.4)
Requirement already satisfied: gitdb<5,>=4.0.1 in /usr/local/lib/python3.10/dist-p
ackages (from gitpython>=3.1.30->-r https://raw.githubusercontent.com/ultralytics/
yolov5/master/requirements.txt (line 5)) (4.0.11)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-
packages (from matplotlib>=3.3->-r https://raw.githubusercontent.com/ultralytics/y
olov5/master/requirements.txt (line 6)) (1.2.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-pack
ages (from matplotlib>=3.3->-r https://raw.githubusercontent.com/ultralytics/yolov
5/master/requirements.txt (line 6)) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist
-packages (from matplotlib>=3.3->-r https://raw.githubusercontent.com/ultralytics/
yolov5/master/requirements.txt (line 6)) (4.53.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist
```

```
-packages (from matplotlib>=3.3->-r https://raw.githubusercontent.com/ultralytics/
yolov5/master/requirements.txt (line 6)) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-p
ackages (from matplotlib>=3.3->-r https://raw.githubusercontent.com/ultralytics/yo
lov5/master/requirements.txt (line 6)) (24.1)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-
packages (from matplotlib>=3.3->-r https://raw.githubusercontent.com/ultralytics/y
olov5/master/requirements.txt (line 6)) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/d
ist-packages (from matplotlib>=3.3->-r https://raw.githubusercontent.com/ultralyti
cs/yolov5/master/requirements.txt (line 6)) (2.8.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.
10/dist-packages (from requests>=2.32.0->-r https://raw.githubusercontent.com/ultr
alytics/yolov5/master/requirements.txt (line 12)) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-pack
ages (from requests>=2.32.0->-r https://raw.githubusercontent.com/ultralytics/yolo
v5/master/requirements.txt (line 12)) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dis
t-packages (from requests>=2.32.0->-r https://raw.githubusercontent.com/ultralytic
s/yolov5/master/requirements.txt (line 12)) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dis
t-packages (from requests>=2.32.0->-r https://raw.githubusercontent.com/ultralytic
s/yolov5/master/requirements.txt (line 12)) (2024.7.4)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages
(from torch>=1.8.0->-r https://raw.githubusercontent.com/ultralytics/yolov5/maste
r/requirements.txt (line 15)) (3.15.4)
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.
10/dist-packages (from torch>=1.8.0->-r https://raw.githubusercontent.com/ultralyt
ics/yolov5/master/requirements.txt (line 15)) (4.12.2)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (f
rom torch>=1.8.0->-r https://raw.githubusercontent.com/ultralytics/yolov5/master/r
equirements.txt (line 15)) (1.13.1)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages
(from torch>=1.8.0->-r https://raw.githubusercontent.com/ultralytics/yolov5/maste
r/requirements.txt (line 15)) (3.3)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages
(from torch>=1.8.0->-r https://raw.githubusercontent.com/ultralytics/yolov5/maste
r/requirements.txt (line 15)) (3.1.4)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages
(from torch>=1.8.0->-r https://raw.githubusercontent.com/ultralytics/yolov5/maste
r/requirements.txt (line 15)) (2024.6.1)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.1.105 in /usr/local/lib/
python3.10/dist-packages (from torch>=1.8.0->-r https://raw.githubusercontent.com/
ultralytics/yolov5/master/requirements.txt (line 15)) (12.1.105)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.1.105 in /usr/local/li
b/python3.10/dist-packages (from torch>=1.8.0->-r https://raw.githubusercontent.co
m/ultralytics/yolov5/master/requirements.txt (line 15)) (12.1.105)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.1.105 in /usr/local/lib/
python3.10/dist-packages (from torch>=1.8.0->-r https://raw.githubusercontent.com/
ultralytics/yolov5/master/requirements.txt (line 15)) (12.1.105)
Requirement already satisfied: nvidia-cudnn-cu12==8.9.2.26 in /usr/local/lib/pytho
n3.10/dist-packages (from torch>=1.8.0->-r https://raw.githubusercontent.com/ultra
lytics/yolov5/master/requirements.txt (line 15)) (8.9.2.26)
Requirement already satisfied: nvidia-cublas-cu12==12.1.3.1 in /usr/local/lib/pyth
on3.10/dist-packages (from torch>=1.8.0->-r https://raw.githubusercontent.com/ultr
alytics/yolov5/master/requirements.txt (line 15)) (12.1.3.1)
Requirement already satisfied: nvidia-cufft-cu12==11.0.2.54 in /usr/local/lib/pyth
on3.10/dist-packages (from torch>=1.8.0->-r https://raw.githubusercontent.com/ultr
alytics/yolov5/master/requirements.txt (line 15)) (11.0.2.54)
Requirement already satisfied: nvidia-curand-cu12==10.3.2.106 in /usr/local/lib/py
thon3.10/dist-packages (from torch>=1.8.0->-r https://raw.githubusercontent.com/ul
tralytics/yolov5/master/requirements.txt (line 15)) (10.3.2.106)
Requirement already satisfied: nvidia-cusolver-cu12==11.4.5.107 in /usr/local/lib/
python3.10/dist-packages (from torch>=1.8.0->-r https://raw.githubusercontent.com/
```

```
ultralytics/yolov5/master/requirements.txt (line 15)) (11.4.5.107)
Requirement already satisfied: nvidia-cusparse-cu12==12.1.0.106 in /usr/local/lib/
python3.10/dist-packages (from torch>=1.8.0->-r https://raw.githubusercontent.com/
ultralytics/yolov5/master/requirements.txt (line 15)) (12.1.0.106)
Requirement already satisfied: nvidia-nccl-cu12==2.20.5 in /usr/local/lib/python3.
10/dist-packages (from torch>=1.8.0->-r https://raw.githubusercontent.com/ultralyt
ics/yolov5/master/requirements.txt (line 15)) (2.20.5)
Requirement already satisfied: nvidia-nvtx-cu12==12.1.105 in /usr/local/lib/python
3.10/dist-packages (from torch>=1.8.0->-r https://raw.githubusercontent.com/ultral
ytics/yolov5/master/requirements.txt (line 15)) (12.1.105)
Requirement already satisfied: triton==2.3.1 in /usr/local/lib/python3.10/dist-pac
kages (from torch>=1.8.0->-r https://raw.githubusercontent.com/ultralytics/yolov5/
master/requirements.txt (line 15)) (2.3.1)
Requirement already satisfied: nvidia-nvjitlink-cu12 in /usr/local/lib/python3.10/
dist-packages (from nvidia-cusolver-cu12==11.4.5.107->torch>=1.8.0->-r https://ra
w.githubusercontent.com/ultralytics/yolov5/master/requirements.txt (line 15)) (12.
6.20)
Requirement already satisfied: py-cpuinfo in /usr/local/lib/python3.10/dist-packag
es (from ultralytics>=8.2.34->-r https://raw.githubusercontent.com/ultralytics/yol
ov5/master/requirements.txt (line 18)) (9.0.0)
Requirement already satisfied: ultralytics-thop>=2.0.0 in /usr/local/lib/python3.1
0/dist-packages (from ultralytics>=8.2.34->-r https://raw.githubusercontent.com/ul
tralytics/yolov5/master/requirements.txt (line 18)) (2.0.0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-pack
ages (from pandas>=1.1.4->-r https://raw.githubusercontent.com/ultralytics/yolov5/
master/requirements.txt (line 27)) (2024.1)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-pa
ckages (from pandas>=1.1.4->-r https://raw.githubusercontent.com/ultralytics/yolov
5/master/requirements.txt (line 27)) (2024.1)
Requirement already satisfied: smmap<6,>=3.0.1 in /usr/local/lib/python3.10/dist-p
ackages (from gitdb<5,>=4.0.1->gitpython>=3.1.30->-r https://raw.githubuserconten
t.com/ultralytics/yolov5/master/requirements.txt (line 5)) (5.0.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages
(from python-dateutil>=2.7->matplotlib>=3.3->-r https://raw.githubusercontent.com/
ultralytics/yolov5/master/requirements.txt (line 6)) (1.16.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-p
ackages (from jinja2->torch>=1.8.0->-r https://raw.githubusercontent.com/ultralyti
cs/yolov5/master/requirements.txt (line 15)) (2.1.5)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.10/dis
t-packages (from sympy->torch>=1.8.0->-r https://raw.githubusercontent.com/ultraly
tics/yolov5/master/requirements.txt (line 15)) (1.3.0)
Downloading thop-0.1.1.post2209072238-py3-none-any.whl (15 kB)
Installing collected packages: thop
Successfully installed thop-0.1.1.post2209072238
```

In [ ]:
```
#mount the drive
from google.colab import drive
drive.mount("/content/gdrive", force_remount=True)
```

```
Mounted at /content/gdrive
```

In [ ]:
```
from PIL import Image
image1 = Image.open('/content/gdrive/MyDrive/test_images/dog_with_person.png')
image2 = Image.open('/content/gdrive/MyDrive/test_images/dog_with_elevator.jpg')
image3 = Image.open('/content/gdrive/MyDrive/test_images/dog_with_tree.jpg')
image4 = Image.open('/content/gdrive/MyDrive/test_images/dog1_1.jpg')
image5 = Image.open('/content/gdrive/MyDrive/test_images/dog1_2.jpg')
image6 = Image.open('/content/gdrive/MyDrive/test_images/dog1_3.jpg')
```

In [ ]:
```
# Model
model = torch.hub.load("ultralytics/yolov5", "yolov5s")
```

```
Using cache found in /root/.cache/torch/hub/ultralytics_yolov5_master
YOLOv5 🚀 2024-8-12 Python-3.10.12 torch-2.3.1+cu121 CPU

Downloading https://github.com/ultralytics/yolov5/releases/download/v7.0/yolov5s.p
t to yolov5s.pt...
100%|██████████| 14.1M/14.1M [00:00<00:00, 83.5MB/s]

Fusing layers...
YOLOv5s summary: 213 layers, 7225885 parameters, 0 gradients, 16.4 GFLOPs
Adding AutoShape...
```

In [ ]:
```
# Inference
results = model(image1)

results.pandas().xyxy[0]
```

Out[ ]:

	xmin	ymin	xmax	ymax	confidence	class	name
0	70.396828	1028.920654	656.174255	2302.261719	0.866599	0	person
1	392.082458	1403.504028	848.679199	2290.931885	0.674810	77	teddy bear
2	394.483215	1431.344116	835.672546	2297.853271	0.506323	16	dog

In [ ]:
```
boxes_df = results.pandas().xyxy[0]
```

In [ ]:
```
# Extract all data so we can use them later
xmin_values = boxes_df['xmin']
ymin_values = boxes_df['ymin']
xmax_values = boxes_df['xmax']
ymax_values = boxes_df['ymax']
names = boxes_df['name']
classes = boxes_df['class']
confidence = boxes_df['confidence']
```

In [ ]:
```
# Filter the DataFrame to get the row(s) where the name is 'dog'
dog_boxes = boxes_df[boxes_df['name'] == 'dog']

# We would expect only one 'dog' object:
if not dog_boxes.empty:

  # More than 1 dog
  # if len(dog_boxes) > 1:
  #   print("Warning: More than one 'dog' object detected.")
  #   index = 0

  #   for row in dog_boxes.iterrows():
  #     xmin = row['xmin']
  #     ymin = row['ymin']
  #     xmax = row['xmax']
  #     ymax = row['ymax']
  #     index += 1

  #   print(f"Dog {index} coordinates: xmin={xmin}, ymin={ymin}, xmax={xmax}, ymax=

  # Only 1 dog case

    xmin = dog_boxes.iloc[0]['xmin']
    ymin = dog_boxes.iloc[0]['ymin']
    xmax = dog_boxes.iloc[0]['xmax']
    ymax = dog_boxes.iloc[0]['ymax']
    print(f"Dog coordinates: xmin={xmin}, ymin={ymin}, xmax={xmax}, ymax={ymax}")
```

```
else:
    print("No 'dog' object detected.")
```

```
Dog coordinates: xmin=394.48321533203125, ymin=1431.3441162109375, xmax=835.672546
3867188, ymax=2297.853271484375
```

In [ ]:
```
# Now that we have the coordniates of the dog, we crop the image first before sendi
image1 = Image.open('/content/gdrive/MyDrive/test_images/dog_with_person.png')
image1 = image1.crop((xmin, ymin, xmax, ymax))
```

In [ ]:
```
display(image1)
```

In [ ]:
```
# From the aobve we see that our dog has been correctly croped out. Now we pass it
# Because we made this part of the code sepreate from the above section now we rewr
```

In [ ]:
```
num_classes = 50
alexnet = models.alexnet(pretrained=True)

#freeze model parameters, those pretrained alexnet parameters won't be updated duri
for param in alexnet.parameters():
  param.requires_grad = False
```

```
# We are adding final layers to AlexNet to fit our tasks for Transfer Learning
# In this case, since we are using small net data with only 5 classes,
# we are using small hidden layer size of 512 and drop out rate
# of 0.5 to make the model more accurate

# The structre of the modyfied alexnet is refernced to this following article:
# https://github.com/kipkoechjosh/APS360-Dog-Breed-Classifier/blob/main/final_dogs_
# After careful tunning of the neruals of each layer we decided to keep the followi

alexnet.classifier.add_module("4", nn.Linear(4096, 512))
alexnet.classifier.add_module("6", nn.Dropout(p=0.5, inplace=False))
alexnet.classifier.add_module("7", nn.Linear(512, 256))
alexnet.classifier.add_module("8", nn.ReLU(inplace=True))
alexnet.classifier.add_module("9", nn.Linear(256, num_classes))
alexnet
```

Out[ ]:
```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=Fals
e)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=Fals
e)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=Fals
e)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=512, bias=True)
    (5): ReLU(inplace=True)
    (6): Dropout(p=0.5, inplace=False)
    (7): Linear(in_features=512, out_features=256, bias=True)
    (8): ReLU(inplace=True)
    (9): Linear(in_features=256, out_features=50, bias=True)
  )
)
```

In [ ]:
```
cd /content/gdrive/MyDrive/aps360largetest/Images
```

```
/content/gdrive/.shortcut-targets-by-id/1fLRn_-T5O6uh9f4YOAyHJSmQT7pYg_6y/Images
```

In [ ]:
```
# Define the paths
large_test_folder = '.'  # Current directory
large_test_train_folder = 'train_3'
large_test_val_folder = 'val_3'
large_test_test_folder = 'test_3'
```

In [ ]:
```
large_test_folder = '/content/gdrive/MyDrive/aps360largetest/Images'
large_test_train_folder = '/content/gdrive/MyDrive/aps360largetest/Images/train_3'
```

```
large_test_val_folder = '/content/gdrive/MyDrive/aps360largetest/Images/val_3'
large_test_test_folder = '/content/gdrive/MyDrive/aps360largetest/Images/test_3'
```

In [ ]:
```
first_time = False
num_images_per_class = 100
num_classes = 50
number_of_selected_class = 50
batch_size = 327
```

In [ ]:
```
# Re split the data in another folder
if first_time:

  # Get all class folders, excluding "train", "valid", "test", "train_2", "valid_2'
  all_classes = [f for f in os.listdir(large_test_folder) if os.path.isdir(os.path.

  # Randomly select 50 classes
  selected_classes = random.sample(all_classes, number_of_selected_class)

  print("Splitting Images into 70% training, 15% validation, 15% testing")
  for class_folder in selected_classes:
      class_path = os.path.join(large_test_folder, class_folder)

      # Create corresponding class folders in train, val, test
      os.makedirs(os.path.join(large_test_train_folder, class_folder), exist_ok=Tru
      os.makedirs(os.path.join(large_test_val_folder, class_folder), exist_ok=True)
      os.makedirs(os.path.join(large_test_test_folder, class_folder), exist_ok=True

      # Get the list of image files for this class
      image_files = [f for f in os.listdir(class_path) if os.path.isfile(os.path.jo

      # Select up to 100 random images from the class
      num_images_to_select = min(num_images_per_class, len(image_files))
      selected_images = random.sample(image_files, num_images_to_select)

      # Split into train and temp (test + val)
      train_files, temp_files = train_test_split(selected_images, test_size=0.3, ra

      # Split temp into val and test
      val_files, test_files = train_test_split(temp_files, test_size=0.5, random_st

      # Copy the files to their respective folders
      for file in train_files:
          shutil.copy(os.path.join(class_path, file), os.path.join(large_test_train
      for file in val_files:
          shutil.copy(os.path.join(class_path, file), os.path.join(large_test_val_f
      for file in test_files:
          shutil.copy(os.path.join(class_path, file), os.path.join(large_test_test_

  first_time = False

else:
  print("Images Already Split")
```

```
Images Already Split
```

In [ ]:
```
# Now we count to check if the data is correct
def count_images(folder):
    count = 0
    for class_folder in os.listdir(folder):
        class_path = os.path.join(folder, class_folder)
        if os.path.isdir(class_path):
            count += len([f for f in os.listdir(class_path) if os.path.isfile(os.pa
    return count
```

```
train_count = count_images(large_test_train_folder)
val_count = count_images(large_test_val_folder)
test_count = count_images(large_test_test_folder)


total_count = train_count + val_count + test_count

print("Number of training images:", train_count, "({:.2f}%)".format(train_count/tot
print("Number of validation images:", val_count, "({:.2f}%)".format(val_count/total
print("Number of testing images:", test_count, "({:.2f}%)".format(test_count/total_
```

```
Number of training images: 3500 (70.00%)
Number of validation images: 750 (15.00%)
Number of testing images: 750 (15.00%)
```

In [ ]:
```
# Define data transformations
data_transform = transforms.Compose([
    transforms.Resize((224, 224)),  # Resize to match CNN input
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])  # Normalize us
])

# Load training data
train_dataset = datasets.ImageFolder(root='train_3', transform=data_transform)
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)

# Load validation data
val_dataset = datasets.ImageFolder(root='val_3', transform=data_transform)
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)

# Load test data
test_dataset = datasets.ImageFolder(root='test_3', transform=data_transform)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
```

In [ ]:
```
# now tring batch size = 327, learning rate = 0.0018
use_cuda = True
torch.cuda.empty_cache()
model_alex = alexnet
model_name = "alexnet"


if use_cuda and torch.cuda.is_available():
  alexnet.cuda()
  model_alex.cuda()
  print('CUDA is available!  Training on GPU ...')
else:
  print('CUDA is not available.  Training on CPU ...')

# train model
train(model_alex, model_name,learning_rate = 0.00018, batch_size=327, num_epochs=5)
```

```
CUDA is not available.  Training on CPU ...
Using alexnet
Iteration:  1 Progress:   1.82  % Time Elapsed:  63.18 s
Iteration:  2 Progress:   3.64  % Time Elapsed:  123.48 s
Iteration:  3 Progress:   5.45  % Time Elapsed:  186.09 s
Iteration:  4 Progress:   7.27  % Time Elapsed:  245.79 s
Iteration:  5 Progress:   9.09  % Time Elapsed:  305.03 s
Iteration:  6 Progress:  10.91  % Time Elapsed:  365.87 s
Iteration:  7 Progress:  12.73  % Time Elapsed:  425.54 s
Iteration:  8 Progress:  14.55  % Time Elapsed:  484.90 s
Iteration:  9 Progress:  16.36  % Time Elapsed:  544.56 s
Iteration:  10 Progress:  18.18  % Time Elapsed:  604.04 s
Iteration:  11 Progress:  20.00  % Time Elapsed:  655.50 s
Epoch 0 Finished.  Time per Epoch:  655.51 s
Iteration:  12 Progress:  21.82  % Time Elapsed:  713.98 s
Iteration:  13 Progress:  23.64  % Time Elapsed:  772.38 s
Iteration:  14 Progress:  25.45  % Time Elapsed:  830.96 s
Iteration:  15 Progress:  27.27  % Time Elapsed:  889.76 s
Iteration:  16 Progress:  29.09  % Time Elapsed:  949.02 s
Iteration:  17 Progress:  30.91  % Time Elapsed:  1008.27 s
Iteration:  18 Progress:  32.73  % Time Elapsed:  1067.11 s
Iteration:  19 Progress:  34.55  % Time Elapsed:  1125.79 s
Iteration:  20 Progress:  36.36  % Time Elapsed:  1184.98 s
Iteration:  21 Progress:  38.18  % Time Elapsed:  1245.01 s
Iteration:  22 Progress:  40.00  % Time Elapsed:  1297.13 s
Epoch 1 Finished.  Time per Epoch:  648.57 s
Iteration:  23 Progress:  41.82  % Time Elapsed:  1357.72 s
Iteration:  24 Progress:  43.64  % Time Elapsed:  1418.14 s
Iteration:  25 Progress:  45.45  % Time Elapsed:  1478.68 s
Iteration:  26 Progress:  47.27  % Time Elapsed:  1538.95 s
Iteration:  27 Progress:  49.09  % Time Elapsed:  1599.47 s
Iteration:  28 Progress:  50.91  % Time Elapsed:  1659.66 s
Iteration:  29 Progress:  52.73  % Time Elapsed:  1720.35 s
Iteration:  30 Progress:  54.55  % Time Elapsed:  1781.08 s
Iteration:  31 Progress:  56.36  % Time Elapsed:  1841.56 s
Iteration:  32 Progress:  58.18  % Time Elapsed:  1901.40 s
Iteration:  33 Progress:  60.00  % Time Elapsed:  1953.04 s
Epoch 2 Finished.  Time per Epoch:  651.01 s
Iteration:  34 Progress:  61.82  % Time Elapsed:  2012.17 s
Iteration:  35 Progress:  63.64  % Time Elapsed:  2071.76 s
Iteration:  36 Progress:  65.45  % Time Elapsed:  2132.15 s
Iteration:  37 Progress:  67.27  % Time Elapsed:  2192.99 s
Iteration:  38 Progress:  69.09  % Time Elapsed:  2254.15 s
Iteration:  39 Progress:  70.91  % Time Elapsed:  2315.19 s
Iteration:  40 Progress:  72.73  % Time Elapsed:  2375.86 s
Iteration:  41 Progress:  74.55  % Time Elapsed:  2437.18 s
Iteration:  42 Progress:  76.36  % Time Elapsed:  2498.18 s
Iteration:  43 Progress:  78.18  % Time Elapsed:  2559.32 s
Iteration:  44 Progress:  80.00  % Time Elapsed:  2610.96 s
Epoch 3 Finished.  Time per Epoch:  652.74 s
Iteration:  45 Progress:  81.82  % Time Elapsed:  2669.62 s
Iteration:  46 Progress:  83.64  % Time Elapsed:  2728.76 s
Iteration:  47 Progress:  85.45  % Time Elapsed:  2788.24 s
Iteration:  48 Progress:  87.27  % Time Elapsed:  2847.26 s
Iteration:  49 Progress:  89.09  % Time Elapsed:  2906.23 s
Iteration:  50 Progress:  90.91  % Time Elapsed:  2965.22 s
Iteration:  51 Progress:  92.73  % Time Elapsed:  3024.96 s
Iteration:  52 Progress:  94.55  % Time Elapsed:  3084.32 s
Iteration:  53 Progress:  96.36  % Time Elapsed:  3143.23 s
Iteration:  54 Progress:  98.18  % Time Elapsed:  3202.00 s
Iteration:  55 Progress:  100.00  % Time Elapsed:  3253.44 s
Epoch 4 Finished.  Time per Epoch:  650.69 s
Final Training Accuracy: 0.4054285714285714
```

```
Final Validation Accuracy: 0.30666666666666664
Total time:   3253.44 s  Time per Epoch:  650.69 s
```

In [ ]:
```
test_accuracy = get_accuracy_of_transfer(model_alex, test_loader)
print("Test Accuracy: {:.2f}%".format(test_accuracy * 100))
```

```
Test Accuracy: 40.71%
```

In [ ]:
```
#Trying Batch size 256
use_cuda = True
torch.cuda.empty_cache()
batch_size = 256
model_alex = alexnet
model_name = "alexnet"


if use_cuda and torch.cuda.is_available():
  alexnet.cuda()
  model_alex.cuda()
  print('CUDA is available!  Training on GPU ...')
else:
  print('CUDA is not available.  Training on CPU ...')

# train model
train(model_alex, model_name, batch_size=batch_size, num_epochs=10)
```

```
CUDA is not available.  Training on CPU ...
Using alexnet
Iteration:  1 Progress:   0.91  % Time Elapsed:  67.74 s
Iteration:  2 Progress:   1.82  % Time Elapsed:  125.97 s
Iteration:  3 Progress:   2.73  % Time Elapsed:  184.52 s
Iteration:  4 Progress:   3.64  % Time Elapsed:  243.13 s
Iteration:  5 Progress:   4.55  % Time Elapsed:  302.60 s
Iteration:  6 Progress:   5.45  % Time Elapsed:  361.88 s
Iteration:  7 Progress:   6.36  % Time Elapsed:  421.12 s
Iteration:  8 Progress:   7.27  % Time Elapsed:  480.04 s
Iteration:  9 Progress:   8.18  % Time Elapsed:  539.56 s
Iteration:  10 Progress:   9.09  % Time Elapsed:  600.09 s
Iteration:  11 Progress:  10.00  % Time Elapsed:  650.97 s
Epoch 0 Finished.  Time per Epoch:  650.97 s
Iteration:  12 Progress:  10.91  % Time Elapsed:  708.91 s
Iteration:  13 Progress:  11.82  % Time Elapsed:  770.46 s
Iteration:  14 Progress:  12.73  % Time Elapsed:  829.11 s
Iteration:  15 Progress:  13.64  % Time Elapsed:  888.04 s
Iteration:  16 Progress:  14.55  % Time Elapsed:  947.01 s
Iteration:  17 Progress:  15.45  % Time Elapsed:  1005.67 s
Iteration:  18 Progress:  16.36  % Time Elapsed:  1064.37 s
Iteration:  19 Progress:  17.27  % Time Elapsed:  1122.91 s
Iteration:  20 Progress:  18.18  % Time Elapsed:  1181.80 s
Iteration:  21 Progress:  19.09  % Time Elapsed:  1241.07 s
Iteration:  22 Progress:  20.00  % Time Elapsed:  1293.36 s
Epoch 1 Finished.  Time per Epoch:  646.68 s
Iteration:  23 Progress:  20.91  % Time Elapsed:  1354.71 s
Iteration:  24 Progress:  21.82  % Time Elapsed:  1415.49 s
Iteration:  25 Progress:  22.73  % Time Elapsed:  1476.19 s
Iteration:  26 Progress:  23.64  % Time Elapsed:  1537.23 s
Iteration:  27 Progress:  24.55  % Time Elapsed:  1598.91 s
Iteration:  28 Progress:  25.45  % Time Elapsed:  1659.59 s
Iteration:  29 Progress:  26.36  % Time Elapsed:  1719.82 s
Iteration:  30 Progress:  27.27  % Time Elapsed:  1779.92 s
Iteration:  31 Progress:  28.18  % Time Elapsed:  1840.26 s
Iteration:  32 Progress:  29.09  % Time Elapsed:  1900.71 s
Iteration:  33 Progress:  30.00  % Time Elapsed:  1952.44 s
Epoch 2 Finished.  Time per Epoch:  650.82 s
Iteration:  34 Progress:  30.91  % Time Elapsed:  2012.27 s
Iteration:  35 Progress:  31.82  % Time Elapsed:  2072.56 s
Iteration:  36 Progress:  32.73  % Time Elapsed:  2134.80 s
Iteration:  37 Progress:  33.64  % Time Elapsed:  2196.06 s
Iteration:  38 Progress:  34.55  % Time Elapsed:  2257.20 s
Iteration:  39 Progress:  35.45  % Time Elapsed:  2317.75 s
Iteration:  40 Progress:  36.36  % Time Elapsed:  2378.23 s
Iteration:  41 Progress:  37.27  % Time Elapsed:  2438.81 s
Iteration:  42 Progress:  38.18  % Time Elapsed:  2498.87 s
Iteration:  43 Progress:  39.09  % Time Elapsed:  2559.30 s
Iteration:  44 Progress:  40.00  % Time Elapsed:  2610.98 s
Epoch 3 Finished.  Time per Epoch:  652.75 s
Iteration:  45 Progress:  40.91  % Time Elapsed:  2670.36 s
Iteration:  46 Progress:  41.82  % Time Elapsed:  2730.10 s
Iteration:  47 Progress:  42.73  % Time Elapsed:  2794.50 s
Iteration:  48 Progress:  43.64  % Time Elapsed:  2880.32 s
Iteration:  49 Progress:  44.55  % Time Elapsed:  2951.16 s
Iteration:  50 Progress:  45.45  % Time Elapsed:  3013.10 s
Iteration:  51 Progress:  46.36  % Time Elapsed:  3074.02 s
Iteration:  52 Progress:  47.27  % Time Elapsed:  3134.48 s
Iteration:  53 Progress:  48.18  % Time Elapsed:  3196.19 s
Iteration:  54 Progress:  49.09  % Time Elapsed:  3253.95 s
Iteration:  55 Progress:  50.00  % Time Elapsed:  3306.13 s
Epoch 4 Finished.  Time per Epoch:  661.23 s
Iteration:  56 Progress:  50.91  % Time Elapsed:  3366.06 s
Iteration:  57 Progress:  51.82  % Time Elapsed:  3425.90 s
```

```
Iteration:  58 Progress:  52.73  % Time Elapsed:  3487.29 s
Iteration:  59 Progress:  53.64  % Time Elapsed:  3547.89 s
Iteration:  60 Progress:  54.55  % Time Elapsed:  3608.47 s
Iteration:  61 Progress:  55.45  % Time Elapsed:  3669.16 s
Iteration:  62 Progress:  56.36  % Time Elapsed:  3729.46 s
Iteration:  63 Progress:  57.27  % Time Elapsed:  3790.84 s
Iteration:  64 Progress:  58.18  % Time Elapsed:  3850.71 s
Iteration:  65 Progress:  59.09  % Time Elapsed:  3910.81 s
Iteration:  66 Progress:  60.00  % Time Elapsed:  3961.85 s
Epoch 5 Finished.  Time per Epoch:  660.31 s
Iteration:  67 Progress:  60.91  % Time Elapsed:  4020.38 s
Iteration:  68 Progress:  61.82  % Time Elapsed:  4078.67 s
Iteration:  69 Progress:  62.73  % Time Elapsed:  4137.95 s
Iteration:  70 Progress:  63.64  % Time Elapsed:  4196.97 s
Iteration:  71 Progress:  64.55  % Time Elapsed:  4256.97 s
Iteration:  72 Progress:  65.45  % Time Elapsed:  4316.42 s
Iteration:  73 Progress:  66.36  % Time Elapsed:  4375.55 s
Iteration:  74 Progress:  67.27  % Time Elapsed:  4434.39 s
Iteration:  75 Progress:  68.18  % Time Elapsed:  4493.58 s
Iteration:  76 Progress:  69.09  % Time Elapsed:  4551.67 s
Iteration:  77 Progress:  70.00  % Time Elapsed:  4603.49 s
Epoch 6 Finished.  Time per Epoch:  657.64 s
Iteration:  78 Progress:  70.91  % Time Elapsed:  4662.93 s
Iteration:  79 Progress:  71.82  % Time Elapsed:  4723.53 s
Iteration:  80 Progress:  72.73  % Time Elapsed:  4784.51 s
Iteration:  81 Progress:  73.64  % Time Elapsed:  4844.90 s
Iteration:  82 Progress:  74.55  % Time Elapsed:  4905.47 s
Iteration:  83 Progress:  75.45  % Time Elapsed:  4965.90 s
Iteration:  84 Progress:  76.36  % Time Elapsed:  5026.09 s
Iteration:  85 Progress:  77.27  % Time Elapsed:  5086.46 s
Iteration:  86 Progress:  78.18  % Time Elapsed:  5146.96 s
Iteration:  87 Progress:  79.09  % Time Elapsed:  5207.42 s
Iteration:  88 Progress:  80.00  % Time Elapsed:  5258.68 s
Epoch 7 Finished.  Time per Epoch:  657.34 s
Iteration:  89 Progress:  80.91  % Time Elapsed:  5317.15 s
Iteration:  90 Progress:  81.82  % Time Elapsed:  5375.75 s
Iteration:  91 Progress:  82.73  % Time Elapsed:  5434.88 s
Iteration:  92 Progress:  83.64  % Time Elapsed:  5494.14 s
Iteration:  93 Progress:  84.55  % Time Elapsed:  5553.07 s
Iteration:  94 Progress:  85.45  % Time Elapsed:  5611.86 s
Iteration:  95 Progress:  86.36  % Time Elapsed:  5670.55 s
Iteration:  96 Progress:  87.27  % Time Elapsed:  5729.95 s
Iteration:  97 Progress:  88.18  % Time Elapsed:  5788.62 s
Iteration:  98 Progress:  89.09  % Time Elapsed:  5847.35 s
Iteration:  99 Progress:  90.00  % Time Elapsed:  5898.57 s
Epoch 8 Finished.  Time per Epoch:  655.40 s
Iteration:  100 Progress:  90.91  % Time Elapsed:  5957.69 s
Iteration:  101 Progress:  91.82  % Time Elapsed:  6017.46 s
Iteration:  102 Progress:  92.73  % Time Elapsed:  6077.92 s
Iteration:  103 Progress:  93.64  % Time Elapsed:  6138.51 s
Iteration:  104 Progress:  94.55  % Time Elapsed:  6199.23 s
Iteration:  105 Progress:  95.45  % Time Elapsed:  6259.40 s
Iteration:  106 Progress:  96.36  % Time Elapsed:  6319.79 s
Iteration:  107 Progress:  97.27  % Time Elapsed:  6380.30 s
Iteration:  108 Progress:  98.18  % Time Elapsed:  6440.94 s
Iteration:  109 Progress:  99.09  % Time Elapsed:  6501.16 s
Iteration:  110 Progress:  100.00  % Time Elapsed:  6552.11 s
Epoch 9 Finished.  Time per Epoch:  655.21 s
Final Training Accuracy: 0.7385714285714285
Final Validation Accuracy: 0.476
Total time:   6552.11 s  Time per Epoch:  655.21 s
```

In [ ]:
```
#Trying Batch size 300, learning rate = 0.003
use_cuda = True
```

```
torch.cuda.empty_cache()
model_alex = alexnet
model_name = "alexnet"


if use_cuda and torch.cuda.is_available():
  alexnet.cuda()
  model_alex.cuda()
  print('CUDA is available!  Training on GPU ...')
else:
  print('CUDA is not available.  Training on CPU ...')

# train model
train(model_alex, model_name,learning_rate = 0.003, batch_size=300, num_epochs=5)
```

```
CUDA is not available.  Training on CPU ...
Using alexnet
Iteration:  1 Progress:   1.82  % Time Elapsed:  60.33 s
Iteration:  2 Progress:   3.64  % Time Elapsed:  120.83 s
Iteration:  3 Progress:   5.45  % Time Elapsed:  181.21 s
Iteration:  4 Progress:   7.27  % Time Elapsed:  243.25 s
Iteration:  5 Progress:   9.09  % Time Elapsed:  303.18 s
Iteration:  6 Progress:  10.91  % Time Elapsed:  363.50 s
Iteration:  7 Progress:  12.73  % Time Elapsed:  423.46 s
Iteration:  8 Progress:  14.55  % Time Elapsed:  483.63 s
Iteration:  9 Progress:  16.36  % Time Elapsed:  543.20 s
Iteration:  10 Progress:  18.18  % Time Elapsed:  603.31 s
Iteration:  11 Progress:  20.00  % Time Elapsed:  654.46 s
Epoch 0 Finished.  Time per Epoch:  654.46 s
Iteration:  12 Progress:  21.82  % Time Elapsed:  712.80 s
Iteration:  13 Progress:  23.64  % Time Elapsed:  771.32 s
Iteration:  14 Progress:  25.45  % Time Elapsed:  830.26 s
Iteration:  15 Progress:  27.27  % Time Elapsed:  889.00 s
Iteration:  16 Progress:  29.09  % Time Elapsed:  947.89 s
Iteration:  17 Progress:  30.91  % Time Elapsed:  1006.84 s
Iteration:  18 Progress:  32.73  % Time Elapsed:  1065.88 s
Iteration:  19 Progress:  34.55  % Time Elapsed:  1125.10 s
Iteration:  20 Progress:  36.36  % Time Elapsed:  1184.30 s
Iteration:  21 Progress:  38.18  % Time Elapsed:  1243.14 s
Iteration:  22 Progress:  40.00  % Time Elapsed:  1294.76 s
Epoch 1 Finished.  Time per Epoch:  647.38 s
Iteration:  23 Progress:  41.82  % Time Elapsed:  1353.99 s
Iteration:  24 Progress:  43.64  % Time Elapsed:  1413.86 s
Iteration:  25 Progress:  45.45  % Time Elapsed:  1474.22 s
Iteration:  26 Progress:  47.27  % Time Elapsed:  1534.19 s
Iteration:  27 Progress:  49.09  % Time Elapsed:  1594.43 s
Iteration:  28 Progress:  50.91  % Time Elapsed:  1654.22 s
Iteration:  29 Progress:  52.73  % Time Elapsed:  1714.01 s
Iteration:  30 Progress:  54.55  % Time Elapsed:  1773.59 s
Iteration:  31 Progress:  56.36  % Time Elapsed:  1833.64 s
Iteration:  32 Progress:  58.18  % Time Elapsed:  1893.92 s
Iteration:  33 Progress:  60.00  % Time Elapsed:  1945.88 s
Epoch 2 Finished.  Time per Epoch:  648.63 s
Iteration:  34 Progress:  61.82  % Time Elapsed:  2005.30 s
Iteration:  35 Progress:  63.64  % Time Elapsed:  2064.50 s
Iteration:  36 Progress:  65.45  % Time Elapsed:  2123.43 s
Iteration:  37 Progress:  67.27  % Time Elapsed:  2182.34 s
Iteration:  38 Progress:  69.09  % Time Elapsed:  2241.57 s
Iteration:  39 Progress:  70.91  % Time Elapsed:  2300.97 s
Iteration:  40 Progress:  72.73  % Time Elapsed:  2360.23 s
Iteration:  41 Progress:  74.55  % Time Elapsed:  2419.98 s
Iteration:  42 Progress:  76.36  % Time Elapsed:  2479.58 s
Iteration:  43 Progress:  78.18  % Time Elapsed:  2539.37 s
Iteration:  44 Progress:  80.00  % Time Elapsed:  2592.17 s
Epoch 3 Finished.  Time per Epoch:  648.04 s
Iteration:  45 Progress:  81.82  % Time Elapsed:  2651.06 s
Iteration:  46 Progress:  83.64  % Time Elapsed:  2709.81 s
Iteration:  47 Progress:  85.45  % Time Elapsed:  2768.39 s
Iteration:  48 Progress:  87.27  % Time Elapsed:  2826.91 s
Iteration:  49 Progress:  89.09  % Time Elapsed:  2886.13 s
Iteration:  50 Progress:  90.91  % Time Elapsed:  2952.62 s
Iteration:  51 Progress:  92.73  % Time Elapsed:  3024.40 s
Iteration:  52 Progress:  94.55  % Time Elapsed:  3095.33 s
Iteration:  53 Progress:  96.36  % Time Elapsed:  3160.09 s
Iteration:  54 Progress:  98.18  % Time Elapsed:  3219.13 s
Iteration:  55 Progress:  100.00  % Time Elapsed:  3271.04 s
Epoch 4 Finished.  Time per Epoch:  654.21 s
Final Training Accuracy: 0.6371428571428571
```

```
Final Validation Accuracy: 0.4186666666666667
Total time:   3271.04 s  Time per Epoch:  654.21 s
```

In [ ]:

```
# now tring batch size = 327, learning rate = 0.0008
use_cuda = True
torch.cuda.empty_cache()
model_alex = alexnet
model_name = "alexnet"


if use_cuda and torch.cuda.is_available():
  alexnet.cuda()
  model_alex.cuda()
  print('CUDA is available!  Training on GPU ...')
else:
  print('CUDA is not available.  Training on CPU ...')

# train model
train(model_alex, model_name,learning_rate = 0.0008, batch_size=327, num_epochs=5)
```

```
CUDA is not available.  Training on CPU ...
Using alexnet
Iteration:  1 Progress:   1.82  % Time Elapsed:  63.56 s
Iteration:  2 Progress:   3.64  % Time Elapsed:  128.19 s
Iteration:  3 Progress:   5.45  % Time Elapsed:  192.37 s
Iteration:  4 Progress:   7.27  % Time Elapsed:  256.54 s
Iteration:  5 Progress:   9.09  % Time Elapsed:  322.72 s
Iteration:  6 Progress:  10.91  % Time Elapsed:  394.79 s
Iteration:  7 Progress:  12.73  % Time Elapsed:  460.62 s
Iteration:  8 Progress:  14.55  % Time Elapsed:  519.73 s
Iteration:  9 Progress:  16.36  % Time Elapsed:  578.67 s
Iteration:  10 Progress:  18.18  % Time Elapsed:  637.46 s
Iteration:  11 Progress:  20.00  % Time Elapsed:  690.14 s
Epoch 0 Finished.  Time per Epoch:  690.14 s
Iteration:  12 Progress:  21.82  % Time Elapsed:  750.70 s
Iteration:  13 Progress:  23.64  % Time Elapsed:  811.06 s
Iteration:  14 Progress:  25.45  % Time Elapsed:  871.39 s
Iteration:  15 Progress:  27.27  % Time Elapsed:  931.82 s
Iteration:  16 Progress:  29.09  % Time Elapsed:  992.28 s
Iteration:  17 Progress:  30.91  % Time Elapsed:  1052.69 s
Iteration:  18 Progress:  32.73  % Time Elapsed:  1113.12 s
Iteration:  19 Progress:  34.55  % Time Elapsed:  1173.26 s
Iteration:  20 Progress:  36.36  % Time Elapsed:  1233.03 s
Iteration:  21 Progress:  38.18  % Time Elapsed:  1293.22 s
Iteration:  22 Progress:  40.00  % Time Elapsed:  1344.58 s
Epoch 1 Finished.  Time per Epoch:  672.29 s
Iteration:  23 Progress:  41.82  % Time Elapsed:  1403.86 s
Iteration:  24 Progress:  43.64  % Time Elapsed:  1462.19 s
Iteration:  25 Progress:  45.45  % Time Elapsed:  1520.84 s
Iteration:  26 Progress:  47.27  % Time Elapsed:  1579.35 s
Iteration:  27 Progress:  49.09  % Time Elapsed:  1638.92 s
Iteration:  28 Progress:  50.91  % Time Elapsed:  1697.66 s
Iteration:  29 Progress:  52.73  % Time Elapsed:  1756.58 s
Iteration:  30 Progress:  54.55  % Time Elapsed:  1815.33 s
Iteration:  31 Progress:  56.36  % Time Elapsed:  1875.03 s
Iteration:  32 Progress:  58.18  % Time Elapsed:  1934.49 s
Iteration:  33 Progress:  60.00  % Time Elapsed:  1985.63 s
Epoch 2 Finished.  Time per Epoch:  661.88 s
Iteration:  34 Progress:  61.82  % Time Elapsed:  2045.61 s
Iteration:  35 Progress:  63.64  % Time Elapsed:  2103.40 s
Iteration:  36 Progress:  65.45  % Time Elapsed:  2161.67 s
Iteration:  37 Progress:  67.27  % Time Elapsed:  2219.83 s
Iteration:  38 Progress:  69.09  % Time Elapsed:  2278.16 s
Iteration:  39 Progress:  70.91  % Time Elapsed:  2337.44 s
Iteration:  40 Progress:  72.73  % Time Elapsed:  2396.95 s
Iteration:  41 Progress:  74.55  % Time Elapsed:  2456.90 s
Iteration:  42 Progress:  76.36  % Time Elapsed:  2516.36 s
Iteration:  43 Progress:  78.18  % Time Elapsed:  2576.60 s
Iteration:  44 Progress:  80.00  % Time Elapsed:  2627.71 s
Epoch 3 Finished.  Time per Epoch:  656.93 s
Iteration:  45 Progress:  81.82  % Time Elapsed:  2686.21 s
Iteration:  46 Progress:  83.64  % Time Elapsed:  2744.21 s
Iteration:  47 Progress:  85.45  % Time Elapsed:  2803.16 s
Iteration:  48 Progress:  87.27  % Time Elapsed:  2861.39 s
Iteration:  49 Progress:  89.09  % Time Elapsed:  2919.95 s
Iteration:  50 Progress:  90.91  % Time Elapsed:  2978.94 s
Iteration:  51 Progress:  92.73  % Time Elapsed:  3038.77 s
Iteration:  52 Progress:  94.55  % Time Elapsed:  3098.06 s
Iteration:  53 Progress:  96.36  % Time Elapsed:  3157.74 s
Iteration:  54 Progress:  98.18  % Time Elapsed:  3217.45 s
Iteration:  55 Progress:  100.00  % Time Elapsed:  3268.43 s
Epoch 4 Finished.  Time per Epoch:  653.69 s
Final Training Accuracy: 0.7417142857142857
```

```
Final Validation Accuracy: 0.47733333333333333
Total time:   3268.44 s  Time per Epoch:  653.69 s
```

In [ ]:
```
test_accuracy = get_accuracy_of_transfer(model_alex, test_loader)
print("Test Accuracy: {:.2f}%".format(test_accuracy * 100))
```

```
Test Accuracy: 76.57%
```

In [ ]:
```
# From this point, we see if we pass the test_loader into the get_accuracy_of_trans
# need to do is update the test_loader data with the ones modifyed by croping the d
```

In [ ]:
```
# We first restate the SSD model
# Model
SSD = torch.hub.load("ultralytics/yolov5", "yolov5s")
```

```
Using cache found in /root/.cache/torch/hub/ultralytics_yolov5_master
YOLOv5 🚀 2024-8-12 Python-3.10.12 torch-2.3.1+cu121 CPU

Fusing layers...
YOLOv5s summary: 213 layers, 7225885 parameters, 0 gradients, 16.4 GFLOPs
Adding AutoShape...
```

In [ ]:
```
first_time = False
```

In [ ]:
```
# DO NOT RUN FOR A SECOND TIME
# DO NOT RUN FOR A SECOND TIME
# DO NOT RUN FOR A SECOND TIME

large_test_folder = '/content/gdrive/MyDrive/aps360largetest/Images'
test_folder = os.path.join(large_test_folder, 'test_3')
crop_folder = os.path.join(large_test_folder, 'test_3_crop')

# Create the new folder if it doesn't exist
if not os.path.exists(crop_folder):
    os.makedirs(crop_folder)

# Iterate through subfolders in the test folder
for dog_class in os.listdir(test_folder):
    class_path = os.path.join(test_folder, dog_class)
    crop_class_path = os.path.join(crop_folder, dog_class)

    # Create the corresponding subfolder in the crop folder
    if not os.path.exists(crop_class_path):
        os.makedirs(crop_class_path)

    # Iterate through images in the class subfolder
    for image_file in os.listdir(class_path):
        image_path = os.path.join(class_path, image_file)
        crop_image_path = os.path.join(crop_class_path, image_file)

        # Load the image, apply modifications, and save to the crop folder
        image = Image.open(image_path)

        # Cropping the image
        ssd_results = SSD(image)
        boxes_df = ssd_results.pandas().xyxy[0]
        # Filter the DataFrame to get the row(s) where the name is 'dog'
        dog_boxes = boxes_df[boxes_df['name'] == 'dog']
        # We would expect only one 'dog' object:
        if not dog_boxes.empty:
          xmin = dog_boxes.iloc[0]['xmin']
          ymin = dog_boxes.iloc[0]['ymin']
          xmax = dog_boxes.iloc[0]['xmax']
          ymax = dog_boxes.iloc[0]['ymax']
```

```
            image = image.crop((xmin, ymin, xmax, ymax))

        image.save(crop_image_path)
```

In [ ]:
```
# Now we load the test data
# Define data transformations
data_transform = transforms.Compose([
  transforms.Resize((224, 224)), # Resize to match CNN input
  transforms.ToTensor(),
  transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5]) # Normalize using
])

# Load test data
test_dataset = datasets.ImageFolder(root='test_3', transform=data_transform)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
```

In [ ]:
```
# Now we see how our model improves after the crop using SSD
test_accuracy = get_accuracy_of_transfer(model_alex, test_loader)
print("Test Accuracy: {:.2f}%".format(test_accuracy * 100))
```

```
Test Accuracy: 85.69%
```

In [ ]:
```
# We see now that with the 2 step pipeline by cropping the image first and then pas
# to achieve an accuracy of 85.69%, which is consderably good performance
```

In [ ]:
```
# We now test to see if we can accuratly predict image1 which was dog with person
# Define data transformations
data_transform = transforms.Compose([
    transforms.Resize((224, 224)), # Resize to match CNN input
    transforms.ToTensor(),
    transforms.Lambda(lambda image: image[:3, :, :]), # Select only the first 3 cha
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5]) # Normalize usi
])
```

In [ ]:
```
image1 = Image.open('/content/gdrive/MyDrive/test_images/dog_with_person.png')
image2 = Image.open('/content/gdrive/MyDrive/test_images/dog_with_elevator.jpg')
image3 = Image.open('/content/gdrive/MyDrive/test_images/dog_with_tree.jpg')
image4 = Image.open('/content/gdrive/MyDrive/test_images/dog1_1.jpg')
image5 = Image.open('/content/gdrive/MyDrive/test_images/dog1_2.jpg')
image6 = Image.open('/content/gdrive/MyDrive/test_images/dog1_3.jpg')
```

In [ ]:
```
# Cropping the image
ssd_results = SSD(image1)
boxes_df = ssd_results.pandas().xyxy[0]
# Filter the DataFrame to get the row(s) where the name is 'dog'
dog_boxes = boxes_df[boxes_df['name'] == 'dog']
# We would expect only one 'dog' object:
if not dog_boxes.empty:
  xmin = dog_boxes.iloc[0]['xmin']
  ymin = dog_boxes.iloc[0]['ymin']
  xmax = dog_boxes.iloc[0]['xmax']
  ymax = dog_boxes.iloc[0]['ymax']
  image1 = image1.crop((xmin, ymin, xmax, ymax))

  display(image1)

  image1_tensor = data_transform(image1)
  image1_tensor = image1_tensor.unsqueeze(0)

with torch.no_grad():
  output = model_alex(image1_tensor)
  _, predicted = torch.max(output, 1)
```

```
print("Predicted class:", predicted.item())
```

```
Predicted class: 47
```

In [ ]:
```
# And we know from before that class 47 was indeed 'n02113712-miniature_poodle', wh
```

In [ ]:
```
# Now we want to try with some other images we collected from internet
# Here we use the dog dataset by TsingHua University of China: https://cg.cs.tsingh

# As names of the TsingHua data set may vary with the Stanford data set, we will ch
```

In [ ]:
```
cd /content/gdrive/MyDrive/extra_online_images
```

```
/content/gdrive/.shortcut-targets-by-id/1uxki2mHAnhqTUe6CiIclMcqGukq65Alx/extra_on
line_images
```

In [ ]:
```
# Now, we uploaded a foder containing 200 samples of miniature_poodle from the Tsin
# The TsingHua Dataset names it with "200-n000010-miniature_poodle", now we change
```

In [ ]:
```
# DO NOT RUN FOR A SECOND TIME
# DO NOT RUN FOR A SECOND TIME
# DO NOT RUN FOR A SECOND TIME
from PIL import Image

large_test_folder = '/content/gdrive/MyDrive/extra_online_images'
test_folder = os.path.join(large_test_folder, 'uncroped')
crop_folder = os.path.join(large_test_folder, 'croped')

# Create the new folder if it doesn't exist
if not os.path.exists(crop_folder):
    os.makedirs(crop_folder)

# Iterate through subfolders in the test folder
for dog_class in os.listdir(test_folder):
    class_path = os.path.join(test_folder, dog_class)
    crop_class_path = os.path.join(crop_folder, dog_class)

    # Create the corresponding subfolder in the crop folder
    if not os.path.exists(crop_class_path):
        os.makedirs(crop_class_path)

    # Iterate through images in the class subfolder
    for image_file in os.listdir(class_path):
        image_path = os.path.join(class_path, image_file)
        crop_image_path = os.path.join(crop_class_path, image_file)

        # Load the image, apply modifications, and save to the crop folder
        image = Image.open(image_path)

        # Cropping the image
        ssd_results = SSD(image)
        boxes_df = ssd_results.pandas().xyxy[0]
        # Filter the DataFrame to get the row(s) where the name is 'dog'
        dog_boxes = boxes_df[boxes_df['name'] == 'dog']
        # We would expect only one 'dog' object:
        if not dog_boxes.empty:
          xmin = dog_boxes.iloc[0]['xmin']
          ymin = dog_boxes.iloc[0]['ymin']
          xmax = dog_boxes.iloc[0]['xmax']
          ymax = dog_boxes.iloc[0]['ymax']
          image = image.crop((xmin, ymin, xmax, ymax))

        image.save(crop_image_path)
```

In [ ]:
```
test_folder = '/content/gdrive/MyDrive/extra_online_images'
```

In [ ]:
```
cd /content/gdrive/MyDrive/extra_online_images
```

```
/content/gdrive/.shortcut-targets-by-id/1uxki2mHAnhqTUe6CiIclMcqGukq65Alx/extra_on
line_images
```

In [ ]:
```
# data transformations to crop the images into same size of 224*224
data_transform = transforms.Compose([
    transforms.Resize((224, 224)),  # Resize to match CNN input
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])  # Normalize us
])


# Load test data
test_dataset = datasets.ImageFolder(root='croped', transform=data_transform)
test_loader = DataLoader(test_dataset, batch_size=327, shuffle=False)
```

In [ ]:
```
accuracy = get_accuracy_of_transfer(model_alex, test_loader)
print("Test Accuracy: {:.2f}%".format(accuracy * 100))
```

```
Test Accuracy: 75.5%
```

In [ ]:
```
alexnet = models.alexnet(pretrained=True)
resnet = models.resnet50(pretrained=True)
wide_resnet = models.wide_resnet50_2(pretrained=True)
effifient_net = models.efficientnet_b0(pretrained=True)
inceptionV3 = models.inception_v3(pretrained=True)
vgg16 = models.vgg16(pretrained=True)
```

```
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: UserWarn
ing: The parameter 'pretrained' is deprecated since 0.13 and may be removed in the
future, please use 'weights' instead.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarn
ing: Arguments other than a weight enum or `None` for 'weights' are deprecated sin
ce 0.13 and may be removed in the future. The current behavior is equivalent to pa
ssing `weights=AlexNet_Weights.IMAGENET1K_V1`. You can also use `weights=AlexNet_W
eights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/alexnet-owt-7be5be79.pth" to /ro
ot/.cache/torch/hub/checkpoints/alexnet-owt-7be5be79.pth
100%|██████████| 233M/233M [00:01<00:00, 135MB/s]
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarn
ing: Arguments other than a weight enum or `None` for 'weights' are deprecated sin
ce 0.13 and may be removed in the future. The current behavior is equivalent to pa
ssing `weights=ResNet50_Weights.IMAGENET1K_V1`. You can also use `weights=ResNet50
_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet50-0676ba61.pth" to /roo
t/.cache/torch/hub/checkpoints/resnet50-0676ba61.pth
100%|██████████| 97.8M/97.8M [00:00<00:00, 134MB/s]
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarn
ing: Arguments other than a weight enum or `None` for 'weights' are deprecated sin
ce 0.13 and may be removed in the future. The current behavior is equivalent to pa
ssing `weights=Wide_ResNet50_2_Weights.IMAGENET1K_V1`. You can also use `weights=W
ide_ResNet50_2_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/wide_resnet50_2-95faca4d.pth" to
/root/.cache/torch/hub/checkpoints/wide_resnet50_2-95faca4d.pth
100%|██████████| 132M/132M [00:01<00:00, 126MB/s]
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarn
ing: Arguments other than a weight enum or `None` for 'weights' are deprecated sin
ce 0.13 and may be removed in the future. The current behavior is equivalent to pa
ssing `weights=EfficientNet_B0_Weights.IMAGENET1K_V1`. You can also use `weights=E
fficientNet_B0_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/efficientnet_b0_rwightman-7f5810
bc.pth" to /root/.cache/torch/hub/checkpoints/efficientnet_b0_rwightman-7f5810bc.p
th
100%|██████████| 20.5M/20.5M [00:00<00:00, 74.0MB/s]
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarn
ing: Arguments other than a weight enum or `None` for 'weights' are deprecated sin
ce 0.13 and may be removed in the future. The current behavior is equivalent to pa
ssing `weights=Inception_V3_Weights.IMAGENET1K_V1`. You can also use `weights=Ince
ption_V3_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/inception_v3_google-0cc3c7bd.pt
h" to /root/.cache/torch/hub/checkpoints/inception_v3_google-0cc3c7bd.pth
100%|██████████| 104M/104M [00:01<00:00, 95.8MB/s]
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarn
ing: Arguments other than a weight enum or `None` for 'weights' are deprecated sin
ce 0.13 and may be removed in the future. The current behavior is equivalent to pa
ssing `weights=VGG16_Weights.IMAGENET1K_V1`. You can also use `weights=VGG16_Weigh
ts.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/vgg16-397923af.pth" to /root/.ca
che/torch/hub/checkpoints/vgg16-397923af.pth
100%|██████████| 528M/528M [00:06<00:00, 85.1MB/s]
```

In [ ]:
```
use_cuda = True
torch.cuda.empty_cache()
```

```
import time
```

In [ ]:
```
# Alexnet
start_time = time.time()

# Train Alexnet model
train(alexnet, "model_alex" , batch_size=327, num_epochs=10)
alex_accuracy = get_accuracy_of_transfer(alexnet, test_loader)
print("Test Accuracy: {:.2f}%".format(alex_accuracy * 100))

end_time = time.time()
elapsed_time = end_time - start_time
print("Time spent: {:.2f} seconds".format(elapsed_time))
```

```
Test Accuracy: 57.12%
Time spent: 2038.12 seconds
```

In [ ]:
```
# ResNet
start_time = time.time()

# Train ResNet model
train(resnet, "resnet", batch_size=327, num_epochs=10)
resnet_accuracy = get_accuracy_of_transfer(resnet, test_loader)
print("ResNet Test Accuracy: {:.2f}%".format(resnet_accuracy * 100))

end_time = time.time()
elapsed_time = end_time - start_time
print("ResNet Time spent: {:.2f} seconds".format(elapsed_time))
```

```
Test Accuracy: 58.47%
Time spent: 4543.70 seconds
```

In [ ]:
```
# Wide ResNet
start_time = time.time()

# Train Wide ResNet model
train(wide_resnet, "wide_resnet", batch_size=327, num_epochs=10)
wide_resnet_accuracy = get_accuracy_of_transfer(wide_resnet, test_loader)
print("Wide ResNet Test Accuracy: {:.2f}%".format(wide_resnet_accuracy * 100))

end_time = time.time()
elapsed_time = end_time - start_time
print("Wide ResNet Time spent: {:.2f} seconds".format(elapsed_time))
```

```
Test Accuracy: 13.63%
Time spent: 5021.05 seconds
```

In [ ]:
```
# EfficientNet
start_time = time.time()

# Train EfficientNet model
train(effifient_net, "efficient_net", batch_size=327, num_epochs=10)
efficient_net_accuracy = get_accuracy_of_transfer(effifient_net, test_loader)
print("EfficientNet Test Accuracy: {:.2f}%".format(efficient_net_accuracy * 100))

end_time = time.time()
elapsed_time = end_time - start_time
print("EfficientNet Time spent: {:.2f} seconds".format(elapsed_time))
```

```
Test Accuracy: 29.85%
Time spent: 3501.44 seconds
```

In [ ]:
```
# InceptionV3
start_time = time.time()
```

```
# Train inceptionV3 model
train(inceptionV3, "inceptionV3", batch_size=327, num_epochs=10)
inceptionV3_accuracy = get_accuracy_of_transfer(inceptionV3, test_loader)
print("InceptionV3 Test Accuracy: {:.2f}%".format(inceptionV3_accuracy * 100))

end_time = time.time()
elapsed_time = end_time - start_time
print("InceptionV3 Time spent: {:.2f} seconds".format(elapsed_time))
```

```
Test Accuracy: 34.21%
Time spent: 4012.20 seconds
```