Practical 2

Big Data Analytics

2CS702

Mistry Unnat

20BCE515

Department of Computer Science and Engineering

Institute of Technology

Nirma University

Ahmedabad

AIM :
Learning limitation of data analytics by applying Machine Learning Techniques on large amount of data. Write a program to read data set from any online website, excel file and CSV file and to perform
a) Linear regression and logistic regression on iris dataset.
b) K-means clustering.
Students will learn the limitation of platform and algorithm

Import Libraries

In [2]:

```
import numpy as np
import pandas as pd
from sklearn import datasets
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import sklearn.metrics as sm
from sklearn.cluster import KMeans
import seaborn as sns
```

Read dataset from csv file

In [3]:

```
load_from_csv = pd.read_csv("iris_dataset.csv")
```

Read dataset from excel file

In [4]:

```
load_from_excel = pd.read_excel("iris_dataset.xlsx")
```

Load DataSet

In [5]:

```
data=datasets.load_iris()
X=pd.DataFrame(data.data, columns=['Sepal Length','Sepal Width', 'Petal Length','Petal W
idth'])
y=pd.DataFrame(data.target, columns=['Target'])
X.head()
```

Out[5]:

	Sepal Length	Sepal Width	Petal Length	Petal Width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

In [6]:

```
y.head()
```

Out[6]:

	Target
0	0

	Target
1	0
2	0
3	0
4	0

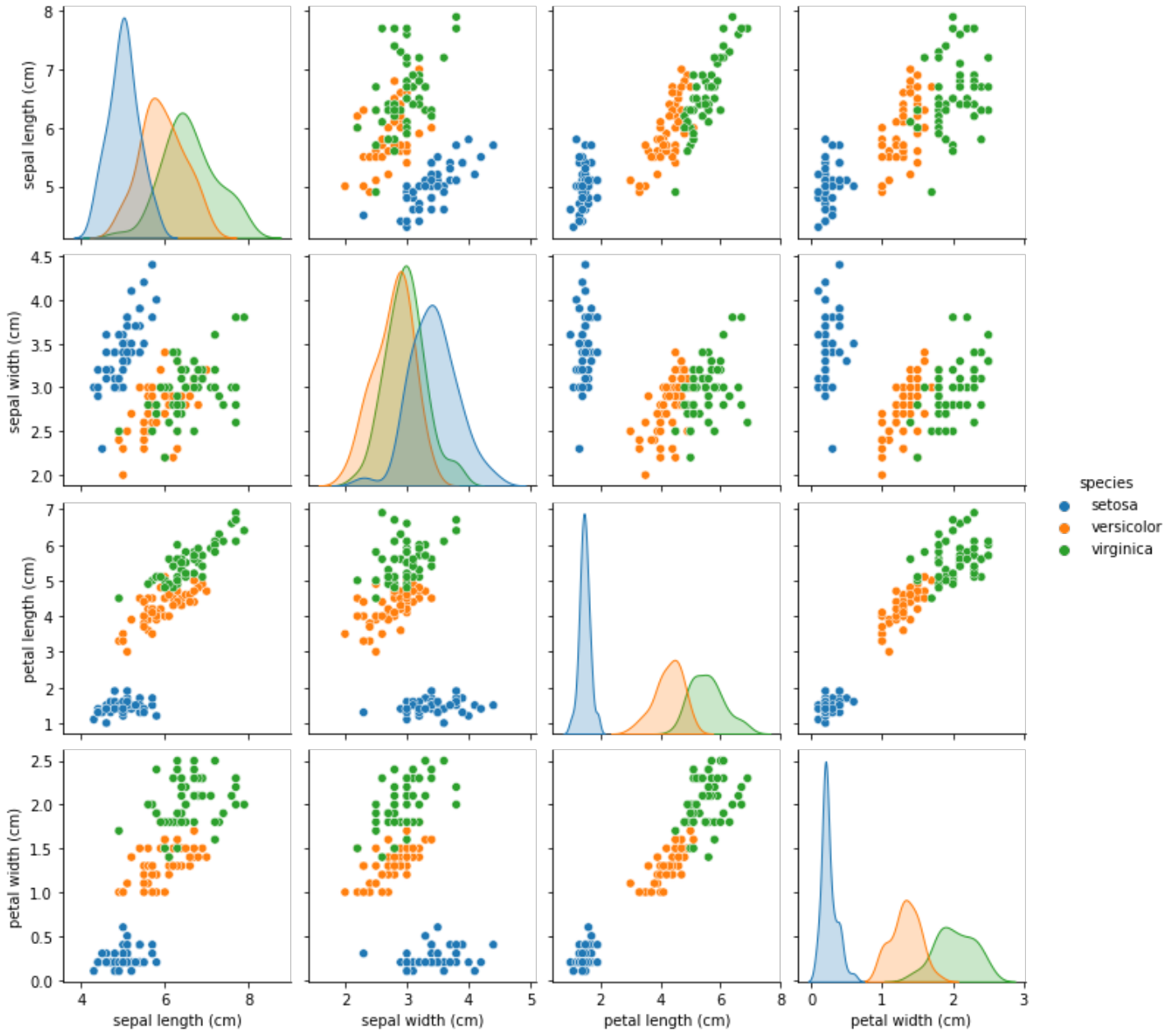
Plot dataset

In [7]:

```
iris_df = pd.DataFrame(data= data.data, columns= data.feature_names)
target_df = pd.DataFrame(data= data.target, columns= ['species'])
def converter(specie):
    if specie == 0:
        return 'setosa'
    elif specie == 1:
        return 'versicolor'
    else:
        return 'virginica'
target_df['species'] = target_df['species'].apply(converter)
# Concatenate the DataFrames
iris_df = pd.concat([iris_df, target_df], axis= 1)
sns.pairplot(iris_df, hue= 'species')
```

Out[7]:

```
<seaborn.axisgrid.PairGrid at 0x20d4a629c40>
```



Linear Regression

In [8]:

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import accuracy_score
lr = LinearRegression()
lr.fit(X,y);
# y_pred=classifier.predict(X)
r2_score = lr.score(X,y)
print("Accuracy : ", r2_score*100,'%')
```

```
Accuracy :  93.03939218549564 %
```

Logistic Regression

In [10]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
classifier=LogisticRegression(random_state=0, max_iter=10000).fit(X,np.ravel(y))
y_pred=classifier.predict(X)

print("Accuracy : " ,accuracy_score(y,y_pred)*100,"%")
```

```
Accuracy :  97.33333333333334 %
```

K means

In [27]:

```
# using KMeans clustering
from sklearn.cluster import KMeans
import numpy as np
from sklearn.metrics import accuracy_score

kmeans = KMeans(n_clusters=4, random_state=42,init='k-means++').fit(X, y)

y_pred = kmeans.predict(X)
print("Accuracy : ",accuracy_score(y,y_pred)*100,"%")
```

```
Accuracy :  70.0 %
```

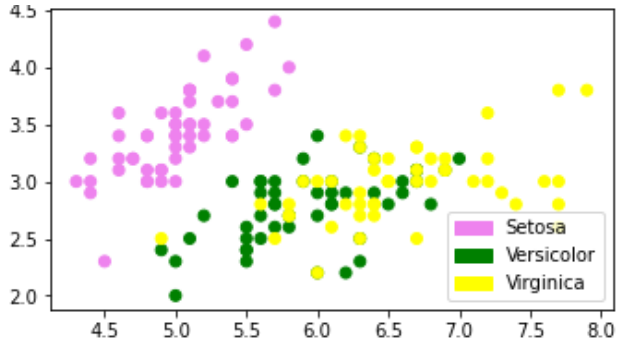
Plotting the graph

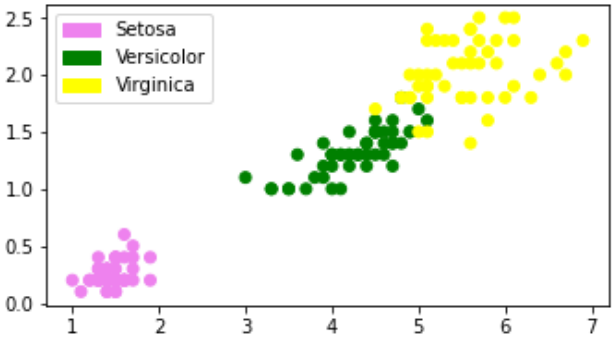
In [28]:

```
plt.figure(figsize=(12,3))
colors = np.array(['violet', 'green', 'yellow'])
iris_targets_legend = np.array(data.target_names)
red_patch = mpatches.Patch(color='violet', label='Setosa')
green_patch = mpatches.Patch(color='green', label='Versicolor')
blue_patch = mpatches.Patch(color='yellow', label='Virginica')


plt.subplot(1, 2, 1)
plt.scatter(X['Sepal Length'], X['Sepal Width'], c=colors[y['Target']])
plt.title('Sepal Length vs Sepal Width')
plt.legend(handles=[red_patch, green_patch, blue_patch])

plt.subplot(1,2,2)
plt.scatter(X['Petal Length'], X['Petal Width'], c= colors[y['Target']])
plt.title('Petal Length vs Petal Width')
plt.legend(handles=[red_patch, green_patch, blue_patch])
```

Out[28]:

```
<matplotlib.legend.Legend at 0x20d4bfdd8b0>
```





Fit k-means model with 4 clusters

In [29]:

```
iris_k_mean_model = KMeans(n_clusters=4)
iris_k_mean_model.fit(X)
```

Out[29]:

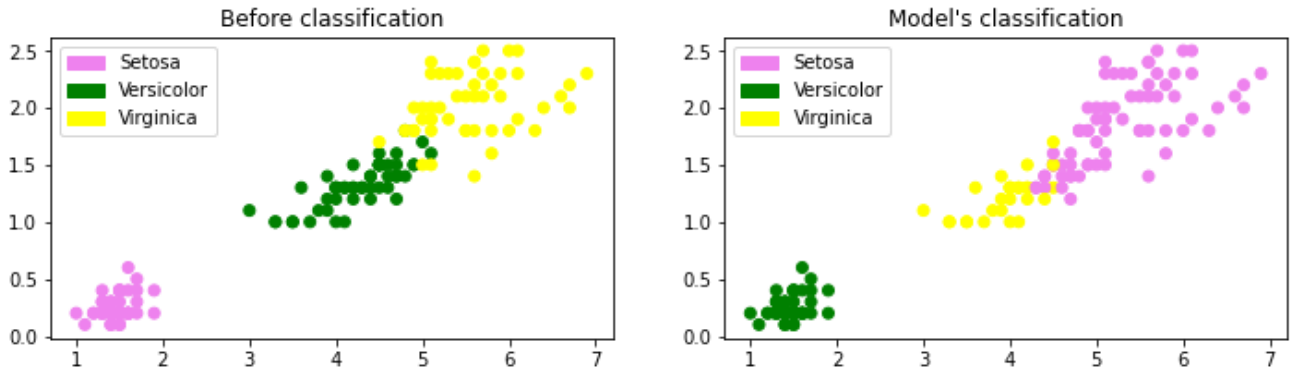
```
KMeans(n_clusters=4)
```

In [35]:

```
plt.figure(figsize=(12,3))

colors = np.array(['violet', 'green', 'yellow'])

predictedY = np.choose(iris_k_mean_model.labels_, [1, 0, 2,0]).astype(np.int64)

plt.subplot(1, 2, 1)
plt.scatter(X['Petal Length'], X['Petal Width'], c=colors[y['Target']])
plt.title('Before classification')
plt.legend(handles=[red_patch, green_patch, blue_patch])

plt.subplot(1, 2, 2)
plt.scatter(X['Petal Length'], X['Petal Width'], c=colors[predictedY])
plt.title("Model's classification")
plt.legend(handles=[red_patch, green_patch, blue_patch])
```

Out[35]:

```
<matplotlib.legend.Legend at 0x20d4eafd910>
```



Working with Large Dataset

Time before training the model

In [55]:

```
from datetime import datetime

now = datetime.now()

current_time = now.strftime("%H:%M:%S")
```

```
print("Current Time =", current_time)
```

```
Current Time = 19:22:20
```

Download required packages and import libraries

In [58]:

```
!pip install torch
!pip install torchvision
import torch
from torch import nn
from torch import optim
from torch.autograd import Variable
from torch.utils.data import DataLoader
from torchvision import datasets, transforms, models
```

```
Requirement already satisfied: torch in c:\users\unnat\anaconda3\lib\site-packages (1.12.
1)
Requirement already satisfied: typing-extensions in c:\users\unnat\anaconda3\lib\site-pac
kages (from torch) (4.1.1)
Collecting torchvision
  Downloading torchvision-0.13.1-cp39-cp39-win_amd64.whl (1.1 MB)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in c:\users\unnat\anaconda3\lib\site
-packages (from torchvision) (9.0.1)
Requirement already satisfied: numpy in c:\users\unnat\anaconda3\lib\site-packages (from
torchvision) (1.21.5)
Requirement already satisfied: torch==1.12.1 in c:\users\unnat\anaconda3\lib\site-package
s (from torchvision) (1.12.1)
Requirement already satisfied: requests in c:\users\unnat\anaconda3\lib\site-packages (fr
om torchvision) (2.27.1)
Requirement already satisfied: typing-extensions in c:\users\unnat\anaconda3\lib\site-pac
kages (from torchvision) (4.1.1)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\users\unnat\anaconda3\lib\site
-packages (from requests->torchvision) (1.26.9)
Requirement already satisfied: charset-normalizer~=2.0.0 in c:\users\unnat\anaconda3\lib\
site-packages (from requests->torchvision) (2.0.4)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\unnat\anaconda3\lib\site-pa
ckages (from requests->torchvision) (2021.10.8)
Requirement already satisfied: idna<4,>=2.5 in c:\users\unnat\anaconda3\lib\site-packages
(from requests->torchvision) (3.3)
Installing collected packages: torchvision
Successfully installed torchvision-0.13.1
```

Train the model on large dataset

In [61]:

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

train_transform = transforms.Compose([transforms.Resize(255),
                                      transforms.CenterCrop(224),
                                      transforms.RandomHorizontalFlip(),
                                      transforms.RandomRotation(20),
                                      transforms.ToTensor(),
                                      transforms.Normalize(mean=[0.485, 0.456, 0.406], s
td=[0.229, 0.224, 0.225])])
train_data = datasets.ImageFolder(root="./waste-segregation-image-dataset/Dataset/train",
transform=train_transform)
trainDataLoader = DataLoader(dataset=train_data, batch_size=64, shuffle=True)


test_transform = transforms.Compose([transforms.Resize(255),
                                      transforms.CenterCrop(224),
                                      transforms.ToTensor(),
                                      transforms.Normalize(mean=[0.485, 0.456, 0.406], s
td=[0.229, 0.224, 0.225])])
test_data = datasets.ImageFolder(root="./waste-segregation-image-dataset/Dataset/val", tr
ansform=test_transform)
testDataLoader = DataLoader(dataset=test_data, batch_size=32, shuffle=True)
```

```
# Load Pre-trained model
model = models.resnet50(pretrained=True)

print("Before: \n", model, "\n\n")

# Freeze our Feature parameters of the model
# Model Parameters: Pre-Learned Weights & Biases
for param in model.parameters():
    # Don't allow gradients/features to update.
    param.requires_grad = False

# Replace the Classification layer with our custom classification layer
from collections import OrderedDict

# Define a sequence of custom layers
# Pass in a OrderedDict to name each of these layers and corresponding functions
classifier = nn.Sequential(OrderedDict([
    ('fc1', nn.Linear(2048, 512)),
    ('relu', nn.ReLU()),
    ('dropout', nn.Dropout(p=0.2)),
    ('fc2', nn.Linear(512, 2)),
    ('output', nn.LogSoftmax(dim=1))
]))

# Add custom layers to the pre-trained model
model.fc = classifier

print("After: \n", model, "\n")

# Train the model with newly added classifier layer
model.to(device)

# Epochs
epochs = 10

# Loss Criterion
criterion = nn.NLLLoss()

# Optimizer
# Since, we only need to update the parameters for the Model classifier.
# So, we use "model.classifier.parameters()" instead of "model.parameters()".
optimizer = optim.Adam(params=model.fc.parameters(), lr=0.001)

train_losses, test_losses = [], []

print("Starting Model Training...")

# ------------- Training Loop ------------
for e in range(epochs):
    running_loss = 0
    # ------------- Training Loop -----------
    for images, labels in trainDataLoader:
        images, labels = Variable(images), Variable(labels)
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model.forward(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    # ------------ Validation Loop -----------
    else:
        test_loss = 0
        accuracy = 0
        with torch.no_grad():
            model.eval()
            for images, labels in testDataLoader:
                images, labels = Variable(images), Variable(labels)
                images, labels = images.to(device), labels.to(device)
                logps = model.forward(images)
                test_loss += criterion(logps, labels)
```

```
                # Actual Probability distribution from Log Probabilities
                ps = torch.exp(logps)
                # topk: returns K largest elements of the given input tensor along a giv
en dimension
                top_k, top_class = ps.topk(1, dim=1)
                # Check if Predicted Output is equal to the Actual Labels
                equals = top_class == labels.view(*top_class.shape)
                # Calculate Accuracy on the Test Dataset
                accuracy += torch.mean(equals.type(torch.FloatTensor))

        # ** Trun back ON the Dropouts for Model Training **
        model.train()

        # Keep track of Training and Test Loss
        train_losses.append(running_loss / len(trainDataLoader))
        test_losses.append(test_loss / len(testDataLoader))

        print("Epoch: {}/{}".format(e + 1, epochs),
              "Training Loss: {:.3f}\t".format(running_loss / len(trainDataLoader)),
              "Test Loss: {:.3f}\t".format(test_loss / len(testDataLoader)),
              "Test Accuracy: {:.3f}\t".format(accuracy / len(testDataLoader)))

print("\nDone...")
```

```
C:\Users\unnat\anaconda3\lib\site-packages\torchvision\models\_utils.py:208: UserWarning:
The parameter 'pretrained' is deprecated since 0.13 and will be removed in 0.15, please u
se 'weights' instead.
  warnings.warn(
C:\Users\unnat\anaconda3\lib\site-packages\torchvision\models\_utils.py:223: UserWarning:
Arguments other than a weight enum or `None` for 'weights' are deprecated since 0.13 and
will be removed in 0.15. The current behavior is equivalent to passing `weights=ResNet50_
Weights.IMAGENET1K_V1`. You can also use `weights=ResNet50_Weights.DEFAULT` to get the mo
st up-to-date weights.
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet50-0676ba61.pth" to C:\Users\unna
t/.cache\torch\hub\checkpoints\resnet50-0676ba61.pth
```

```
Before:
 ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=Tr
ue)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=Fal
se)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=Tr
ue)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=Tr
ue)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=Fal
se)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=Tr
ue)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
```

```
rue)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=Tr
ue)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=Fal
se)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=Tr
ue)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (relu): ReLU(inplace=True)
    )
  )
  (layer2): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=F
alse)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
alse)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
alse)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (relu): ReLU(inplace=True)
    )
    (3): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
alse)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
```

```
rue)
      (relu): ReLU(inplace=True)
    )
  )
  (layer3): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=F
alse)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(512, 1024, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
alse)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
alse)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
      (relu): ReLU(inplace=True)
    )
    (3): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
alse)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
      (relu): ReLU(inplace=True)
    )
    (4): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
alse)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
```

```
True)
      (relu): ReLU(inplace=True)
    )
    (5): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
alse)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
      (relu): ReLU(inplace=True)
    )
  )
  (layer4): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=F
alse)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
alse)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
alse)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
      (relu): ReLU(inplace=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=2048, out_features=1000, bias=True)
)


After:
 ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
```

```
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=Tr
ue)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=Fal
se)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=Tr
ue)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=Tr
ue)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=Fal
se)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=Tr
ue)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=Tr
ue)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=Fal
se)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=Tr
ue)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (relu): ReLU(inplace=True)
    )
  )
  (layer2): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=F
alse)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
```

```
alse)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
alse)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (relu): ReLU(inplace=True)
    )
    (3): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
alse)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (relu): ReLU(inplace=True)
    )
  )
  (layer3): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=F
alse)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(512, 1024, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
alse)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
```

```
alse)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
      (relu): ReLU(inplace=True)
    )
    (3): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
alse)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
      (relu): ReLU(inplace=True)
    )
    (4): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
alse)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
      (relu): ReLU(inplace=True)
    )
    (5): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
alse)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
      (relu): ReLU(inplace=True)
    )
  )
  (layer4): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=F
alse)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
```

```
alse)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
alse)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
      (relu): ReLU(inplace=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Sequential(
    (fc1): Linear(in_features=2048, out_features=512, bias=True)
    (relu): ReLU()
    (dropout): Dropout(p=0.2, inplace=False)
    (fc2): Linear(in_features=512, out_features=2, bias=True)
    (output): LogSoftmax(dim=1)
  )
)

Starting Model Training...

C:\Users\unnat\anaconda3\lib\site-packages\PIL\Image.py:945: UserWarning: Palette images
with Transparency expressed in bytes should be converted to RGBA images
  warnings.warn(

---------------------------------------------------------------------------
OSError                                   Traceback (most recent call last)
Input In [61], in <cell line: 69>()
     70 running_loss = 0
     71 # ------------- Training Loop -----------
---> 72 for images, labels in trainDataLoader:
     73     images, labels = Variable(images), Variable(labels)
     74     images, labels = images.to(device), labels.to(device)

File ~\anaconda3\lib\site-packages\torch\utils\data\dataloader.py:681, in _BaseDataLoader
Iter.__next__(self)
    678 if self._sampler_iter is None:
    679     # TODO(https://github.com/pytorch/pytorch/issues/76750)
    680     self._reset()  # type: ignore[call-arg]
--> 681 data = self._next_data()
    682 self._num_yielded += 1
    683 if self._dataset_kind == _DatasetKind.Iterable and \
    684         self._IterableDataset_len_called is not None and \
    685         self._num_yielded > self._IterableDataset_len_called:

File ~\anaconda3\lib\site-packages\torch\utils\data\dataloader.py:721, in _SingleProcessD
ataLoaderIter._next_data(self)
    719 def _next_data(self):
    720     index = self._next_index()  # may raise StopIteration
--> 721     data = self._dataset_fetcher.fetch(index)  # may raise StopIteration
    722     if self._pin_memory:
    723         data = _utils.pin_memory.pin_memory(data, self._pin_memory_device)

File ~\anaconda3\lib\site-packages\torch\utils\data\_utils\fetch.py:49, in _MapDatasetFet
cher.fetch(self, possibly_batched_index)
     47 def fetch(self, possibly_batched_index):
     48     if self.auto_collation:
---> 49         data = [self.dataset[idx] for idx in possibly_batched_index]
     50     else:
```

```
     51         data = self.dataset[possibly_batched_index]

File ~\anaconda3\lib\site-packages\torch\utils\data\_utils\fetch.py:49, in <listcomp>(.0)
     47 def fetch(self, possibly_batched_index):
     48     if self.auto_collation:
---> 49         data = [self.dataset[idx] for idx in possibly_batched_index]
     50     else:
     51         data = self.dataset[possibly_batched_index]

File ~\anaconda3\lib\site-packages\torchvision\datasets\folder.py:230, in DatasetFolder._
_getitem__(self, index)
    222 """
    223 Args:
    224     index (int): Index
   (...)
    227     tuple: (sample, target) where target is class_index of the target class.
    228 """
    229 path, target = self.samples[index]
--> 230 sample = self.loader(path)
    231 if self.transform is not None:
    232     sample = self.transform(sample)

File ~\anaconda3\lib\site-packages\torchvision\datasets\folder.py:269, in default_loader(
path)
    267     return accimage_loader(path)
    268 else:
--> 269     return pil_loader(path)

File ~\anaconda3\lib\site-packages\torchvision\datasets\folder.py:249, in pil_loader(path
)
    247 with open(path, "rb") as f:
    248     img = Image.open(f)
--> 249     return img.convert("RGB")

File ~\anaconda3\lib\site-packages\PIL\Image.py:889, in Image.convert(self, mode, matrix,
dither, palette, colors)
    847 def convert(self, mode=None, matrix=None, dither=None, palette=WEB, colors=256):
    848     """
    849     Returns a converted copy of this image. For the "P" mode, this
    850     method translates pixels through the palette.  If mode is
   (...)
    886     :returns: An :py:class:`~PIL.Image.Image` object.
    887     """
--> 889     self.load()
    891     has_transparency = self.info.get("transparency") is not None
    892     if not mode and self.mode == "P":
    893         # determine default mode

File ~\anaconda3\lib\site-packages\PIL\ImageFile.py:247, in ImageFile.load(self)
    245         break
    246     else:
--> 247         raise OSError(
    248             "image file is truncated "
    249             f"({len(b)} bytes not processed)"
    250         )
    252 b = b + s
    253 n, err_code = decoder.decode(b)

OSError: image file is truncated (3 bytes not processed)
```

Time after training the model
It took around 12 minutes

In [63]:

```
now = datetime.now()

current_time = now.strftime("%H:%M:%S")
print("Current Time =", current_time)
```

```
Current Time = 19:34:34
```

Conclusion on working on large dataset :
The dataset size is above 1 GB.
The model was training on the dataset without any errors for around 12 minutes.
We got an unexpected error afterwards.
If your system has high specs, it will take comparatively lesser time to train the model.
The time on training the model increases with the size of data.

Time : Large datasets require large amount of time to collect data, process data, annotate data, analyze data, build models.
Storage : Yes, if you are dealing with TB of data, then you have to store it somehow, somewhere and that will not be your typical desktop or laptop. It may be cloud or some additional space for which you have to pay extra dollars.
Money : Besides purchasing additional storage, you may need additional computing resources because your RAM cannot read and process all that huge data. So you need to shell out additional money on purchasing high end (read GPU) devices.
Resources : If the data volume is really large, then one person might not be able to manage it properly and we may need a team of people working on it together. That means, unfortunately, spending more money.
Privacy : If you are dealing with human data, then there is a big ethics issue. In short, you cannot release such data in public domain, not even on public services like cloud.
Missingness : With large data some large problems and missingness is one of them. No data collection mechanism is full-proof and be ready to have strategy in place to deal with missing values.
Mixed Data : The world is not merely numerical, it is categorical, ordinal, textual and everything in between. Be ready to handle mixed data in a real world application.

END