```
# Importing Libraries Used.
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
```

+ Code + Text

1. Convolutional Neural Networks:

```
from __future__ import print_function, division

import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
import torch.backends.cudnn as cudnn
import numpy as np
import torchvision
from torchvision import datasets, models, transforms
import matplotlib.pyplot as plt
import time
import os
import copy
from tensorflow.keras.layers import Dense
from tensorflow.keras.regularizers import l2
cudnn.benchmark = True
plt.ion()   # interactive mode
```

```
!wget https://download.pytorch.org/tutorial/hymenoptera_data.zip
!unzip /content/hymenoptera_data.zip
```

```
--2022-05-07 17:05:37--  https://download.pytorch.org/tutorial/hymenoptera_data.z
Resolving download.pytorch.org (download.pytorch.org)... 18.160.200.71, 18.160.20
Connecting to download.pytorch.org (download.pytorch.org)|18.160.200.71|:443... c
HTTP request sent, awaiting response... 200 OK
Length: 47286322 (45M) [application/zip]
Saving to: ‘hymenoptera_data.zip’

hymenoptera_data.zi 100%[===================>]  45.10M   113MB/s    in 0.4s

2022-05-07 17:05:38 (113 MB/s) - ‘hymenoptera_data.zip’ saved [47286322/47286322]

Archive:  /content/hymenoptera_data.zip
   creating: hymenoptera_data/
   creating: hymenoptera_data/train/
   creating: hymenoptera_data/train/ants/
  inflating: hymenoptera_data/train/ants/0013035.jpg
  inflating: hymenoptera_data/train/ants/1030023514_aad5c608f9.jpg
  inflating: hymenoptera_data/train/ants/1095476100_3906d8afde.jpg
  inflating: hymenoptera_data/train/ants/1099452230_d1949d3250.jpg
  inflating: hymenoptera_data/train/ants/116570827_e9c126745d.jpg
```

```
  inflating: hymenoptera_data/train/ants/1225872729_6f0856588f.jpg
  inflating: hymenoptera_data/train/ants/1262877379_64fcada201.jpg
  inflating: hymenoptera_data/train/ants/1269756697_0bce92cdab.jpg
  inflating: hymenoptera_data/train/ants/1286984635_5119e80de1.jpg
  inflating: hymenoptera_data/train/ants/132478121_2a430adea2.jpg
  inflating: hymenoptera_data/train/ants/1360291657_dc248c5eea.jpg
  inflating: hymenoptera_data/train/ants/1368913450_e146e2fb6d.jpg
  inflating: hymenoptera_data/train/ants/1473187633_63ccaacea6.jpg
  inflating: hymenoptera_data/train/ants/148715752_302c84f5a4.jpg
  inflating: hymenoptera_data/train/ants/1489674356_09d48dde0a.jpg
  inflating: hymenoptera_data/train/ants/149244013_c529578289.jpg
  inflating: hymenoptera_data/train/ants/150801003_3390b73135.jpg
  inflating: hymenoptera_data/train/ants/150801171_cd86f17ed8.jpg
  inflating: hymenoptera_data/train/ants/154124431_65460430f2.jpg
  inflating: hymenoptera_data/train/ants/162603798_40b51f1654.jpg
  inflating: hymenoptera_data/train/ants/1660097129_384bf54490.jpg
  inflating: hymenoptera_data/train/ants/167890289_dd5ba923f3.jpg
  inflating: hymenoptera_data/train/ants/1693954099_46d4c20605.jpg
  inflating: hymenoptera_data/train/ants/175998972.jpg
  inflating: hymenoptera_data/train/ants/178538489_bec7649292.jpg
  inflating: hymenoptera_data/train/ants/1804095607_0341701e1c.jpg
  inflating: hymenoptera_data/train/ants/1808777855_2a895621d7.jpg
  inflating: hymenoptera_data/train/ants/188552436_605cc9b36b.jpg
  inflating: hymenoptera_data/train/ants/1917341202_d00a7f9af5.jpg
  inflating: hymenoptera_data/train/ants/1924473702_daa9aacdbe.jpg
  inflating: hymenoptera_data/train/ants/196057951_63bf063b92.jpg
  inflating: hymenoptera_data/train/ants/196757565_326437f5fe.jpg
  inflating: hymenoptera_data/train/ants/201558278_fe4caecc76.jpg
  inflating: hymenoptera_data/train/ants/201790779_527f4c0168.jpg
  inflating: hymenoptera_data/train/ants/2019439677_2db655d361.jpg
  inflating: hymenoptera_data/train/ants/207947948_3ab29d7207.jpg
  inflating: hymenoptera_data/train/ants/20935278_9190345f6b.jpg
  inflating: hymenoptera_data/train/ants/224655713_3956f7d39a.jpg
  inflating: hymenoptera_data/train/ants/2265824718_2c96f485da.jpg
  inflating: hymenoptera_data/train/ants/2265825502_fff99cfd2d.jpg
  inflating: hymenoptera_data/train/ants/226951206_d6bf946504.jpg
```

```
data_transforms = {
    'train': transforms.Compose([
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
    'val': transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
}

data_dir = '/content/hymenoptera_data'
image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir, x),
                                          data_transforms[x])
                  for x in ['train', 'val']}
```

```
dataloaders = {x: torch.utils.data.DataLoader(image_datasets[x], batch_size=4,
                                              shuffle=True, num_workers=4)
              for x in ['train', 'val']}
dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'val']}
class_names = image_datasets['train'].classes

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

```
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:490: UserWarnin
  cpuset_checked))
```

Visualize a few images

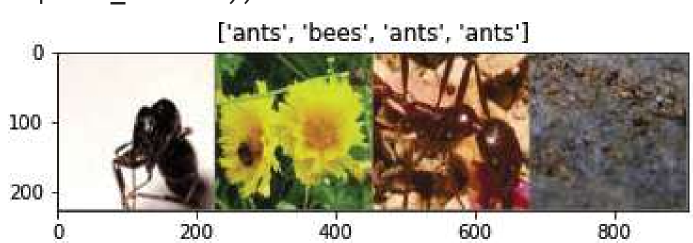
```
def imshow(inp, title=None):
    """Imshow for Tensor."""
    inp = inp.numpy().transpose((1, 2, 0))
    mean = np.array([0.485, 0.456, 0.406])
    std = np.array([0.229, 0.224, 0.225])
    inp = std * inp + mean
    inp = np.clip(inp, 0, 1)
    plt.imshow(inp)
    if title is not None:
        plt.title(title)
    plt.pause(0.001)  # pause a bit so that plots are updated


# Get a batch of training data
inputs, classes = next(iter(dataloaders['train']))

# Make a grid from batch
out = torchvision.utils.make_grid(inputs)

imshow(out, title=[class_names[x] for x in classes])
```

```
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:490: UserWarnin
  cpuset_checked))
```



Training the model

```
def train_model(model, criterion, optimizer, scheduler, num_epochs=25):
```

```
    since = time.time()

    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    for epoch in range(num_epochs):
        print(f'Epoch {epoch}/{num_epochs - 1}')
        print('-' * 10)

        # Each epoch has a training and validation phase
        for phase in ['train', 'val']:
            if phase == 'train':
                model.train()  # Set model to training mode
            else:
                model.eval()   # Set model to evaluate mode

            running_loss = 0.0
            running_corrects = 0

            # Iterate over data.
            for inputs, labels in dataloaders[phase]:
                inputs = inputs.to(device)
                labels = labels.to(device)

                # zero the parameter gradients
                optimizer.zero_grad()

                # forward
                # track history if only in train
                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    _, preds = torch.max(outputs, 1)
                    loss = criterion(outputs, labels)

                    # backward + optimize only if in training phase
                    if phase == 'train':
                        loss.backward()
                        optimizer.step()

                # statistics
                running_loss += loss.item() * inputs.size(0)
                running_corrects += torch.sum(preds == labels.data)
            if phase == 'train':
                scheduler.step()

            epoch_loss = running_loss / dataset_sizes[phase]
            epoch_acc = running_corrects.double() / dataset_sizes[phase]

            print(f'{phase} Loss: {epoch_loss:.4f} Acc: {epoch_acc:.4f}')

            # deep copy the model
            if phase == 'val' and epoch_acc > best_acc:
                best_acc = epoch_acc
                best_model_wts = copy.deepcopy(model.state_dict())
```

```
        print()

    time_elapsed = time.time() - since
    print(f'Training complete in {time_elapsed // 60:.0f}m {time_elapsed % 60:.0f}s')
    print(f'Best val Acc: {best_acc:4f}')

    # load best model weights
    model.load_state_dict(best_model_wts)
    return model
```

Training the model(for plotting)

```
def train_model_plot(model, criterion, optimizer, scheduler, num_epochs):
    since = time.time()

    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0
    epoch_loss=[]
    epoch_acc=[]


    for epoch in range(num_epochs):
        print(f'Epoch {epoch}/{num_epochs - 1}')
        print('-' * 10)

        # Each epoch has a training and validation phase
        for phase in ['train', 'val']:
            if phase == 'train':
                model.train()  # Set model to training mode
            else:
                model.eval()   # Set model to evaluate mode

            running_loss = 0.0
            running_corrects = 0

            # Iterate over data.
            for inputs, labels in dataloaders[phase]:
                inputs = inputs.to(device)
                labels = labels.to(device)

                # zero the parameter gradients
                optimizer.zero_grad()

                # forward
                # track history if only in train
                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    _, preds = torch.max(outputs, 1)
                    loss = criterion(outputs, labels)

                    # backward + optimize only if in training phase
```

```
                if phase == 'train':
                    loss.backward()
                    optimizer.step()

            # statistics
            running_loss += loss.item() * inputs.size(0)
            running_corrects += torch.sum(preds == labels.data)
        if phase == 'train':
            scheduler.step()

        epoch_loss.append((running_loss / dataset_sizes[phase]))
        epoch_acc.append( (running_corrects.double() / dataset_sizes[phase]).cpu())


        # deep copy the model
        if phase == 'val' and epoch_acc[-1] > best_acc:
            best_acc = epoch_acc[-1]
            best_model_wts = copy.deepcopy(model.state_dict())

    print()

time_elapsed = time.time() - since
print(f'Training complete in {time_elapsed // 60:.0f}m {time_elapsed % 60:.0f}s')
print(f'Best val Acc: {best_acc:4f}')

# load best model weights
model.load_state_dict(best_model_wts)
return model,epoch_loss,epoch_acc
```

Visualizing the model predictions

```
def visualize_model(model, num_images=6):
    was_training = model.training
    model.eval()
    images_so_far = 0
    fig=plt.figure()

    with torch.no_grad():
        for i, (inputs, labels) in enumerate(dataloaders['val']):
            inputs = inputs.to(device)
            labels = labels.to(device)

            outputs = model(inputs)
            _, preds = torch.max(outputs, 1)

            for j in range(inputs.size()[0]):
                images_so_far += 1
                ax = plt.subplot(num_images//2, 2, images_so_far)
                ax.axis('off')
                ax.set_title(f'predicted: {class_names[preds[j]]}')
                imshow(inputs.cpu().data[j])
```

```
            if images_so_far == num_images:
                model.train(mode=was_training)
                return
    model.train(mode=was_training)
```

Fine tuning the conv net

```
model_ft = models.resnet18(pretrained=True)
num_ftrs = model_ft.fc.in_features
# Here the size of each output sample is set to 2.
# Alternatively, it can be generalized to nn.Linear(num_ftrs, len(class_names)).
model_ft.fc = nn.Linear(num_ftrs, 2)

model_ft = model_ft.to(device)

criterion = nn.CrossEntropyLoss()

# Observe that all parameters are being optimized
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)

# Decay LR by a factor of 0.1 every 7 epochs
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=7, gamma=0.1)
```

```
Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to /root/.ca
```

100% 44.7M/44.7M [00:00<00:00, 51.2MB/s]

Train and evaluate

```
model_ft = train_model(model_ft, criterion, optimizer_ft, exp_lr_scheduler,
                       num_epochs=25)

visualize_model(model_ft)
```

```
Epoch 0/24
----------
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:490: UserWarnin
  cpuset_checked))
train Loss: 0.5748 Acc: 0.6967
val Loss: 0.2279 Acc: 0.9020

Epoch 1/24
----------
train Loss: 0.4621 Acc: 0.7992
val Loss: 0.2592 Acc: 0.9020

Epoch 2/24
----------
train Loss: 0.4377 Acc: 0.8566
val Loss: 0.6092 Acc: 0.8105

Epoch 3/24
----------
train Loss: 0.6375 Acc: 0.7951
val Loss: 0.3083 Acc: 0.8693

Epoch 4/24
----------
train Loss: 0.3683 Acc: 0.8484
val Loss: 0.3145 Acc: 0.9150

Epoch 5/24
----------
train Loss: 0.5224 Acc: 0.7787
val Loss: 0.3675 Acc: 0.8497

Epoch 6/24
----------
train Loss: 0.4880 Acc: 0.8074
val Loss: 0.3547 Acc: 0.9020

Epoch 7/24
----------
train Loss: 0.3773 Acc: 0.8484
val Loss: 0.1945 Acc: 0.9281
```

Plotting validation loss and accuracy

```
def plot_loss(model,num_epochs):
  x_val=[i+1 for i in range(num_epochs)]
  y_loss=[]
  y_acc=[]
  for i in range(len(model[1])):
    if i%2==0:
      y_loss.append(model[1][i])
  for i in range(len(model[2])):
    if i%2==0:
      y_acc.append(model[2][i])
  plt.rcParams['figure.figsize'] = [9, 9]
```

```
    plt.xticks(x_val, x_val)
    plt.plot(x_val,y_loss)
    plt.plot(x_val,y_acc)
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['loss', 'accuracy'], loc='upper right')
    # plt.show()
```

```
model_ft_update = train_model_plot(model_ft, criterion, optimizer_ft, exp_lr_scheduler,num
plot_loss(model_ft_update,5)
```

```
Epoch 0/4
----------
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:490: UserWarnin
  cpuset_checked))
```

Changing the learning rate, momentum, and number of epochs

```
Epoch 3/4
```

```
def epoch_conv(lr,momentum,num_epochs):
    model_ft = models.resnet18(pretrained=True)
    num_ftrs = model_ft.fc.in_features
    model_ft.fc = nn.Linear(num_ftrs, 2)
    model_ft = model_ft.to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer_ft = optim.SGD(model_ft.parameters(), lr=lr, momentum=momentum)
    exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=7, gamma=0.1)
    model_ft_update = train_model_plot(model_ft, criterion, optimizer_ft, exp_lr_scheduler
    plot_loss(model_ft_update,num_epochs)
    print("convergence for loss and accuracy for learning_rate= {} momentum={}  ".format(l

epoch_conv(0.001,0.5,10)
```

```
Epoch 0/9
----------
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:490: UserWarnir
  cpuset_checked))

Epoch 1/9
----------

Epoch 2/9
----------

Epoch 3/9
----------

Epoch 4/9
----------

Epoch 5/9
----------

Epoch 6/9
----------

Epoch 7/9
----------
```

```
epoch_conv(0.008,0.1,15)
```

```
Epoch 0/14
----------
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:490: UserWarnir
  cpuset_checked))

Epoch 1/14
----------

Epoch 2/14
----------

Epoch 3/14
----------

Epoch 4/14
----------

Epoch 5/14
----------

Epoch 6/14
----------

Epoch 7/14
----------
```

```
epoch_conv(0.01,0.9,20)
```

```
Epoch 0/19
----------
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:490: UserWarni
  cpuset_checked))

Epoch 1/19
----------

Epoch 2/19
----------

Epoch 3/19
----------

Epoch 4/19
----------

Epoch 5/19
----------

Epoch 6/19
----------

Epoch 7/19
----------

Epoch 8/19
----------

Epoch 9/19
----------

Epoch 10/19
----------

Epoch 11/19
----------

Epoch 12/19
----------

Epoch 13/19
----------

Epoch 14/19
----------

Epoch 15/19
----------
```

IIT Bombay

213070024

Home Logout IITB Website

Roll no	213070024	**Name**	Abhishek Verma
Department	Electrical Engineering	**Program**	M.Tech (Control and Computing)

Performance Summary | Graduation Requirements | Personal Information | Forms/Requests | Payment

Academic Performance Summary

Year	Sem	SPI	CPI	Sem Credits Used for SPI	Completed Semester Credits	Cumulative Credits Used for CPI	Completed Cumulative Credits
2021	Spring	8.36	8.19	28.0	28.0	52.0	52.0
2021	Autumn	8.0	8.0	24.0	24.0	24.0	24.0

Semester-wise Details

*This registration is subject to approval(s) from faculty advisor/Course Instructor/Academic office.

Year/Semester: 2022-23/Autumn

Course Code	Course Name	Credits	Tag	Grade	Credit/Audit
SC 649	Embedded Control & Robotics	6.0	Department elective	Not allotted	C

Year/Semester: 2022-23/Project

Course Code	Course Name	Credits	Tag	Grade	Credit/Audit
EE 797	I Stage Project	42.0	Core course	Not allotted	C

Year/Semester: 2021-22/Spring

Course Code	Course Name	Credits	Tag	Grade	Credit/Audit
EE 613	Nonlinear Dynamical Systems	6.0	Core course	AB	C
EE 622	Optimal Control Systems	6.0	Core course	BB	C
EE 636	Matrix Computations	6.0	Core course	BB	C
EE 694	Seminar	4.0	Core course	AB	C
EE 769	Introduction to Machine Learning	6.0	Department elective	BB	C

GC 101	Gender in the workplace	0.0	Core course	PP	N

Year/Semester: 2021-22/Autumn

Course Code	Course Name	Credits	Tag	Grade	Credit/Audit
EE 601	Statistical Signal Analysis	6.0	Core course	BB	C
EE 615	Control and Computational Laboratory	6.0	Core course	AB	C
EE 635	Applied Linear Algebra	6.0	Core course	BC	C
EE 640	Multivariable Control Systems	6.0	Core course	BB	C
EE 899	Communication Skills	6.0	Core course	PP	N