Hands-on Activity 4.2 Transfer Learning on PyTorch	
Course Code: CPE 313	**Program: BSCPE**
Course Title: Advance Machine Learning and Deep Learning	**Date Performed: 02/19/2026**
Section: CPE32S3	**Date Submitted: 02/19/2026**
Name: REUEL CHRISTIAN PORNOBE	**Instructor:** Engr. NEAL BARTON JAMES MATIRA

1. Discussion

Objectives:

This activity aims to demonstrate the use of PyTorch in transfer learning to solve a given problem.

2. Materials and Equipment

- Computer
- Internet

3. Procedure

Follow the instructions as provided in the notebook attached in the instruction materials.
Output:

Provide a lab report showing:

All procedures followed (with screenshots) in the notebookLinks to an external site..
Supplementary Activity:

Answer to the following supplementary activity:

Choose a pretrained model.

Finetune on your dataset from the previous activity.

Evaluate the performance of the previous model to this finetuned model.

Utilize the pretrained ConvNet model as fixed feature extractor.

Evaluate the performance of the previous model to this finetuned model.

Discuss the following:

How did finetuning affect your performance?
Which of the different situations for rule of thumb were applicable to you?

4. Output

Transfer Learning using PyTorch

[1]

```
!nvidia-smi
```

```
Thu Feb 19 05:55:14 2026
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 580.82.07              Driver Version: 580.82.07      CUDA Version: 13.0     |
+-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  Tesla T4                       Off |   00000000:00:04.0 Off |                    0 |
| N/A   52C    P8             11W /   70W |       0MiB /  15360MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI              PID   Type   Process name                        GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
|  No running processes found                                                             |
+-----------------------------------------------------------------------------------------+
```

Load Important Libraries

[2]

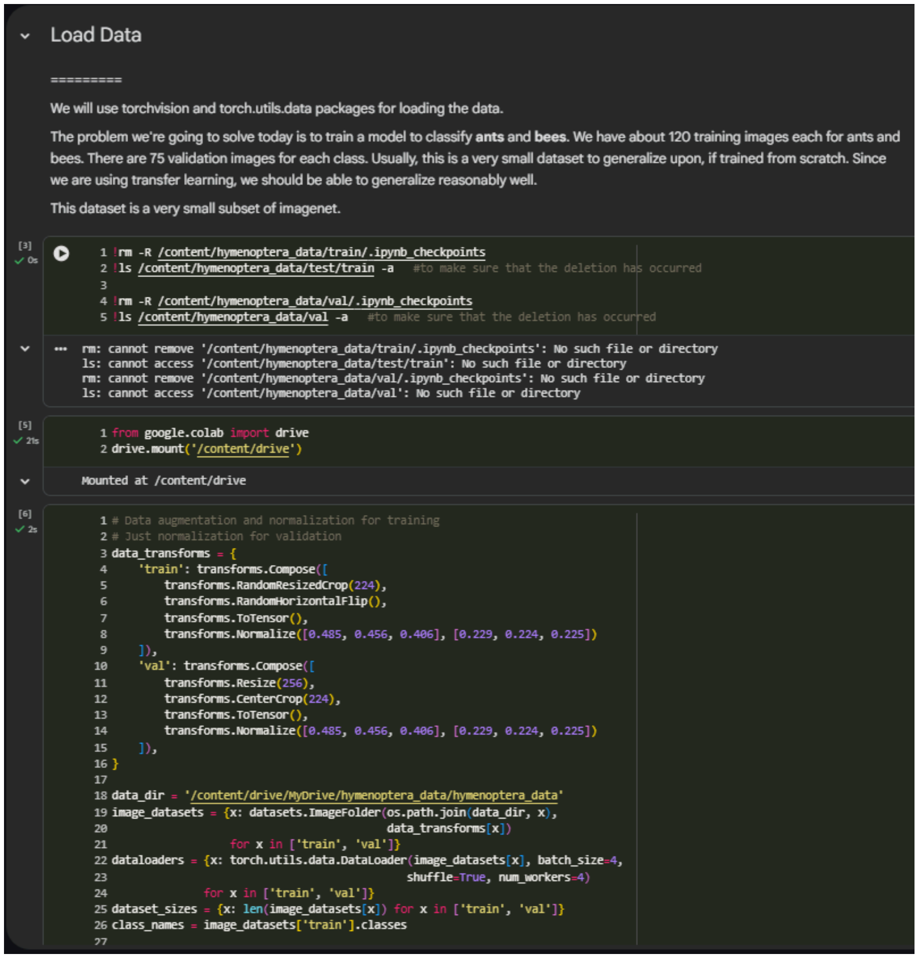
```python
# License: BSD
# Author: Sasank Chilamkurthy

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
from PIL import Image
from tempfile import TemporaryDirectory

cudnn.benchmark = True
plt.ion()   # interactive mode
```

```
<contextlib.ExitStack at 0x7e0de0ff08c0>
```

Importing all the necessary Libraries



Load Data

=========

We will use torchvision and torch.utils.data packages for loading the data.

The problem we're going to solve today is to train a model to classify **ants** and **bees**. We have about 120 training images each for ants and bees. There are 75 validation images for each class. Usually, this is a very small dataset to generalize upon, if trained from scratch. Since we are using transfer learning, we should be able to generalize reasonably well.

This dataset is a very small subset of imagenet.

[3]

```
!rm -R /content/hymenoptera_data/train/.ipynb_checkpoints
!ls /content/hymenoptera_data/test/train -a    #to make sure that the deletion has occurred

!rm -R /content/hymenoptera_data/val/.ipynb_checkpoints
!ls /content/hymenoptera_data/val -a    #to make sure that the deletion has occurred
```

```
rm: cannot remove '/content/hymenoptera_data/train/.ipynb_checkpoints': No such file or directory
ls: cannot access '/content/hymenoptera_data/test/train': No such file or directory
rm: cannot remove '/content/hymenoptera_data/val/.ipynb_checkpoints': No such file or directory
ls: cannot access '/content/hymenoptera_data/val': No such file or directory
```

[5]

```
from google.colab import drive
drive.mount('/content/drive')
```

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

[6]

```
# Data augmentation and normalization for training
# Just normalization for validation
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/drive/MyDrive/hymenoptera_data/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
```

Loading and transforming the data for Resnet

Visualize a few images

======================

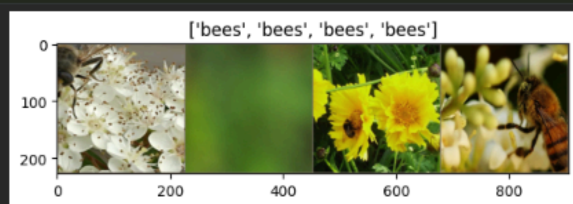
Let's visualize a few training images so as to understand the data augmentations.

```
def imshow(inp, title=None):
    """Display image 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])
```



Created a function to visualize images from the data

Visualizing the model predictions

==================================

Generic function to display predictions for a few images

```
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)
```

Visualize the model prediction and image

Training the model

==================

Now, let's write a general function to train a model. Here, we will illustrate:

- Scheduling the learning rate
- Saving the best model

In the following, parameter `scheduler` is an LR scheduler object from `torch.optim.lr_scheduler`.

[8]

```python
def train_model(model, criterion, optimizer, scheduler, num_epochs=25):
    since = time.time()

    # Create a temporary directory to save training checkpoints
    with TemporaryDirectory() as tempdir:
        best_model_params_path = os.path.join(tempdir, 'best_model_params.pt')

        torch.save(model.state_dict(), best_model_params_path)
        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)
```

Variables Terminal

Creates a function to train a model

Visualizing the model predictions

==================================

Generic function to display predictions for a few images

[9]

```python
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)
```

Finetuning the ConvNet

======================

Load a pretrained model and reset final fully connected layer.

[10]

```python
model_ft = models.resnet18(weights='IMAGENET1K_V1')
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
```

Fine tune the Resnet model for the hymenoptera data

Train and evaluate

==================

It should take around 15-25 min on CPU. On GPU though, it takes less than a minute.

[11]

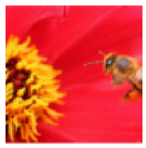
```
model_ft = train_model(model_ft, criterion, optimizer_ft, exp_lr_scheduler,
                       num_epochs=25)
```

Show hidden output

[12]

```
visualize_model(model_ft)
```

predicted: bees



predicted: bees

predicted: bees

Model training of the Fine tuned model

ConvNet as fixed feature extractor

==================================

Here, we need to freeze all the network except the final layer. We need to set `requires_grad = False` to freeze the parameters so that the gradients are not computed in `backward()`.

You can read more about this in the documentation here.

[13]

```
model_conv = torchvision.models.resnet18(weights='IMAGENET1K_V1')
for param in model_conv.parameters():
    param.requires_grad = False

# Parameters of newly constructed modules have requires_grad=True by default
num_ftrs = model_conv.fc.in_features
model_conv.fc = nn.Linear(num_ftrs, 2)

model_conv = model_conv.to(device)

criterion = nn.CrossEntropyLoss()

# Observe that only parameters of final layer are being optimized as
# opposed to before.
optimizer_conv = optim.SGD(model_conv.fc.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_conv, step_size=7, gamma=0.1)
```

Trains only the last layer of the Resnet Model

Choose a pretrained model.

```python
from torchvision.models import inception_v3
supp_model = torchvision.models.inception_v3(weights='IMAGENET1K_V1')
```

```python
# Data for Inception V3
data_transforms = {
    'train': transforms.Compose([
        transforms.RandomResizedCrop(299),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
    'val': transforms.Compose([
        transforms.Resize(299),
        transforms.CenterCrop(299),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
}

data_dir = '/content/drive/MyDrive/hymenoptera_data/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

# We want to be able to train our model on an `accelerator <https://pytorch.org/docs/stable/torch.html#accelerators>`__
# such as CUDA, MPS, MTIA, or XPU. If the current accelerator is available, we will use it. Otherwise, we use the CPU.

device = torch.accelerator.current_accelerator().type if torch.accelerator.is_available() else "cpu"
print(f"Using {device} device")
```

```
Using cuda device
```

In supplementary Activity, I chose Inception_V3 then changed the Data_transform from 254 -> 299.

Finetune on your dataset from the previous activity.

```python
import torchvision
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler

supp_model_ft = torchvision.models.inception_v3(
    weights='IMAGENET1K_V1',
    aux_logits=True
)

# Replace main classifier
num_ftrs = supp_model_ft.fc.in_features
supp_model_ft.fc = nn.Linear(num_ftrs, 2)

# Replace auxiliary classifier
num_ftrs_aux = supp_model_ft.AuxLogits.fc.in_features
supp_model_ft.AuxLogits.fc = nn.Linear(num_ftrs_aux, 2)

supp_model_ft = supp_model_ft.to(device)

criterion = nn.CrossEntropyLoss()

optimizer_ft = optim.SGD(
    supp_model_ft.parameters(),
    lr=0.001,
    momentum=0.9
)

exp_lr_scheduler = lr_scheduler.StepLR(
    optimizer_ft,
    step_size=7,
    gamma=0.1
)
```

Fine tune for Inception V3

Evaluate the performance of the previous model to this finetuned model.

[75]

```python
def train_model_inception_v3(model, criterion, optimizer, scheduler, num_epochs=25):
    since = time.time()

    with TemporaryDirectory() as tempdir:
        best_model_params_path = os.path.join(tempdir, 'best_model_params.pt')
        torch.save(model.state_dict(), best_model_params_path)

        best_acc = 0.0

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

            for phase in ['train', 'val']:
                if phase == 'train':
                    model.train()
                else:
                    model.eval()

                running_loss = 0.0
                running_corrects = 0

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

                    optimizer.zero_grad()

                    with torch.set_grad_enabled(phase == 'train'):
                        outputs = model(inputs)

                        if phase == 'train' and hasattr(outputs, 'aux_logits'):
                            loss1 = criterion(outputs.logits, labels)
                            loss2 = criterion(outputs.aux_logits, labels)
                            loss = loss1 + 0.4 * loss2
                            logits = outputs.logits
                        else:
                            if hasattr(outputs, 'logits'):
                                logits = outputs.logits
                            else:
                                logits = outputs
                            loss = criterion(logits, labels)

                        _, preds = torch.max(logits, 1)

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

                    running_loss += loss.item() * inputs.size(0)
```

[76]

```python
supp_model_ft = train_model_inception_v3(supp_model_ft, criterion, optimizer_ft, exp_lr_scheduler,
                                         num_epochs=25)
```

Show hidden output

Train Inception V3 with a new train function for it.

Utilize the pretrained ConvNet model as fixed feature extractor.

```
import torchvision
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler

model_conv = torchvision.models.inception_v3(
    weights='IMAGENET1K_V1',
    aux_logits=True
)

# Freeze all layers
for param in model_conv.parameters():
    param.requires_grad = False

# Replace main classifier
num_ftrs = model_conv.fc.in_features
model_conv.fc = nn.Linear(num_ftrs, 2)

# Replace auxiliary classifier
num_ftrs_aux = model_conv.AuxLogits.fc.in_features
model_conv.AuxLogits.fc = nn.Linear(num_ftrs_aux, 2)

model_conv = model_conv.to(device)

criterion = nn.CrossEntropyLoss()

# Only optimize the new classifier layers
optimizer_conv = optim.SGD(
    list(model_conv.fc.parameters()) + list(model_conv.AuxLogits.fc.parameters()),
    lr=0.001,
    momentum=0.9
)

exp_lr_scheduler = lr_scheduler.StepLR(
    optimizer_conv,
    step_size=7,
    gamma=0.1
)

```

```
model_conv = train_model_inception_v3(model_conv, criterion, optimizer_conv,
                                      exp_lr_scheduler, num_epochs=25)
```

Trains the Inception V3 with only its last layer.

Procedure Model 1
Training complete in 2m 32s Best val Acc: 0.941176
Procedure Model 2
Training complete in 1m 40s Best val Acc: 0.960784
Supplementary Model 1
Training complete in 3m 52s Best val Acc: 0.9542
Supplementary Model 2

```
Training complete in 2m 41s
Best val Acc: 0.9412
```

5. Supplementary Activity

In supplementary Activity, I chose Inception_V3 then changed the Data_transform from 254 -> 299.

Finetune on your dataset from the previous activity.

[72]

```
import torchvision
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler

supp_model_ft = torchvision.models.inception_v3(
    weights='IMAGENET1K_V1',
    aux_logits=True
)

# Replace main classifier
num_ftrs = supp_model_ft.fc.in_features
supp_model_ft.fc = nn.Linear(num_ftrs, 2)

# Replace auxiliary classifier
num_ftrs_aux = supp_model_ft.AuxLogits.fc.in_features
supp_model_ft.AuxLogits.fc = nn.Linear(num_ftrs_aux, 2)

supp_model_ft = supp_model_ft.to(device)

criterion = nn.CrossEntropyLoss()

optimizer_ft = optim.SGD(
    supp_model_ft.parameters(),
    lr=0.001,
    momentum=0.9
)

exp_lr_scheduler = lr_scheduler.StepLR(
    optimizer_ft,
    step_size=7,
    gamma=0.1
)
```

Fine tune for Inception V3

Evaluate the performance of the previous model to this finetuned model.

```python
def train_model_inception_v3(model, criterion, optimizer, scheduler, num_epochs=25):
    since = time.time()

    with TemporaryDirectory() as tempdir:
        best_model_params_path = os.path.join(tempdir, 'best_model_params.pt')
        torch.save(model.state_dict(), best_model_params_path)

        best_acc = 0.0

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

            for phase in ['train', 'val']:
                if phase == 'train':
                    model.train()
                else:
                    model.eval()

                running_loss = 0.0
                running_corrects = 0

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

                    optimizer.zero_grad()

                    with torch.set_grad_enabled(phase == 'train'):
                        outputs = model(inputs)

                        if phase == 'train' and hasattr(outputs, 'aux_logits'):
                            loss1 = criterion(outputs.logits, labels)
                            loss2 = criterion(outputs.aux_logits, labels)
                            loss = loss1 + 0.4 * loss2
                            logits = outputs.logits
                        else:
                            if hasattr(outputs, 'logits'):
                                logits = outputs.logits
                            else:
                                logits = outputs
                            loss = criterion(logits, labels)

                        _, preds = torch.max(logits, 1)

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

                    running_loss += loss.item() * inputs.size(0)
```

```python
supp_model_ft = train_model_inception_v3(supp_model_ft, criterion, optimizer_ft, exp_lr_scheduler,
                                         num_epochs=25)
```

Show hidden output

Train Inception V3 with a new train function for it.

Utilize the pretrained ConvNet model as fixed feature extractor.

```
[79]
import torchvision
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler

model_conv = torchvision.models.inception_v3(
    weights='IMAGENET1K_V1',
    aux_logits=True
)

# Freeze all layers
for param in model_conv.parameters():
    param.requires_grad = False

# Replace main classifier
num_ftrs = model_conv.fc.in_features
model_conv.fc = nn.Linear(num_ftrs, 2)

# Replace auxiliary classifier
num_ftrs_aux = model_conv.AuxLogits.fc.in_features
model_conv.AuxLogits.fc = nn.Linear(num_ftrs_aux, 2)

model_conv = model_conv.to(device)

criterion = nn.CrossEntropyLoss()

# Only optimize the new classifier layers
optimizer_conv = optim.SGD(
    list(model_conv.fc.parameters()) + list(model_conv.AuxLogits.fc.parameters()),
    lr=0.001,
    momentum=0.9
)

exp_lr_scheduler = lr_scheduler.StepLR(
    optimizer_conv,
    step_size=7,
    gamma=0.1
)
```

```
[80]
model_conv = train_model_inception_v3(model_conv, criterion, optimizer_conv,
                                      exp_lr_scheduler, num_epochs=25)
```

Trains the Inception V3 with only its last layer.

Supplementary Model 1
Training complete in 3m 52s Best val Acc: 0.9542
Supplementary Model 2
Training complete in 2m 41s Best val Acc: 0.9412

6. Conclusion

In this lab activity, I did transfer learning of the Resnet for classifying the dataset, hymenoptera. I learned how it works first hand and how to only change or finetune the last layers of the existing model. For the supplementary activity, I used inception v3 which has different transform size compared to the Resnet model. Therefore, I changed the sizes of data from 254 -> 299. I also

changed how the train_model function works for inception v3. Overall, I was able to understand how to do basic Transfer learning of data.

6. Assessment Rubric

Lab Activity Rubric

Criteria	Ratings						Pts
SO 7 PI 1 **Student Outcome 7.1** Acquire and apply new knowledge from outside sources. threshold: 4.8 pts	6 pts Excellent | Educational interests and pursuits exist and flourish outside classroom requirements,knowledge and/or experiences are pursued independently and applies knowledge learned into practice	5 pts Good | Educational interests and pursuits exist and flourish outside classroom requirements,knowledge and/or experiences are pursued independently	4 pts Satisfactory | Look beyond classroom requirements, showing interest in pursuing knowledge independently	3 pts Unsatisfactory | Begins to look beyond classroom requirements, showing interest in pursuing knowledge independently	2 pts Poor | Relies on classroom instruction only	1 pts Very Poor | No initiative or interest in acquiring new knowledge	6 pts
SO 7 PI 2 **Student Outcome 7.2** Learn independently threshold: 4.8 pts	6 pts Excellent | Completes an assigned task independently and practices continuous improvement	5 pts Good | Completes an assigned task without supervision or guidance	4 pts Satisfactory | Requires minimal guidance to complete an assigned task	3 pts Unsatisfactory | Requires detailed or step-by-step instructions to complete a task	2 pts Poor | Shows little interest to complete a task independently	1 pts Very Poor | No interest to complete a task independently	6 pts
SO 7 PI 3 **Student Outcome 7.3** Critical thinking in the broadest context of technological change threshold: 4.8 pts	6 pts Excellent | Synthesizes and integrates information from a variety of sources; formulates a clear and precise perspective; draws appropriate conclusions	5 pts Good | Evaluate information from a variety of sources; formulates a clear and precise perspective.	4 pts Satisfactory | Analyze information from a variety of sources; formulates a clear and precise perspective.	3 pts Unsatisfactory | Apply the gathered information to formulate the problem	2 pts Poor | Gather and summarized the information from a variety of sources but failed to formulate the problem	1 pts Very Poor | Gather information from a variety of sources	6 pts
SO 7 PI 4 **Student Outcome 7.4** Creativity and adaptability to new and emerging technologies threshold: 4.8 pts	6 pts Excellent | Ideas are combined in original and creative ways in line with the new and emerging technology trends to solve a problem or address an issue.	5 pts Good | Ideas are creative and adapt the new knowledge to solve a problem or address an issue	4 pts Satisfactory | Ideas are creative in solving a problem, or address an issue	3 pts Unsatisfactory | Shows some creative ways to solve the problem	2 pts Poor | Shows initiative and attempt to develop creative ideas to solve the problem	1 pts Very Poor | Ideas are copied or restated from the sources consulted	6 pts
							Total Points: 24