Technological Institute of the Philippines	Quezon City - Computer Engineering
Course Code:	CPE 313
Code Title:	Advanced Machine Learning and Deep Learning
2nd Semester	AY 2025-2026
Assessment Task 4.1	Supplementary Activity on Transfer Learning
Name	Corpuz, Micki Laurren B.
Section	CPE32S3
Date Performed:	23 February 2026
Date Submitted:	23 February 2026
Instructor:	Engr. Neal Barton James Matira

> Instruction:

- Perform Transfer Learning on PyTorch, afterwards perform transfer learning (either fine tuning or a feature extraction) on your chosen dataset.
- Take note of your experiment.
- Reach at least 95% test accuracy.

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
from torchvision import datasets, models, transforms
import matplotlib.pyplot as plt
import os, time, copy

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using:", device)
```

```
Using: cuda
```

```
class DataModule:
    def __init__(self, data_dir, input_size=224, batch_size=32, num_workers=4):
        self.data_dir = data_dir
        self.input_size = input_size
        self.batch_size = batch_size
        self.num_workers = num_workers

        self.data_transforms = {
            'train': transforms.Compose([
                transforms.RandomResizedCrop(input_size),
                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(input_size),
                transforms.ToTensor(),
                transforms.Normalize([0.485,0.456,0.406],
                                     [0.229,0.224,0.225])
            ]),
        }

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

        self.dataloaders = {
            x: torch.utils.data.DataLoader(self.image_datasets[x],
                                           batch_size=batch_size,
                                           shuffle=True,
                                           num_workers=num_workers)
            for x in ['train', 'val']
        }

        self.dataset_sizes = {x: len(self.image_datasets[x]) for x in ['train','val']}
        self.class_names = self.image_datasets['train'].classes
```

```
class ModelMaker:
    def __init__(self, num_classes, feature_extract=False, model_name="resnet18"):
        self.num_classes = num_classes
        self.feature_extract = feature_extract
        self.model_name = model_name

    def create(self):
        model = getattr(models, self.model_name)(weights="IMAGENET1K_V1")

        if self.feature_extract:
            for param in model.parameters():
                param.requires_grad = False

        if "resnet" in self.model_name:
            num_ftrs = model.fc.in_features
            model.fc = nn.Linear(num_ftrs, self.num_classes)

        elif "efficientnet" in self.model_name:
            num_ftrs = model.classifier[1].in_features
            model.classifier[1] = nn.Linear(num_ftrs, self.num_classes)

        return model.to(device)
```

```
class Trainer:
    def __init__(self, model, dataloaders, dataset_sizes, lr=0.001, epochs=25):
        self.model = model
        self.dataloaders = dataloaders
        self.dataset_sizes = dataset_sizes
```

```
        self.epochs = epochs

        self.criterion = nn.CrossEntropyLoss()

        self.optimizer = optim.SGD(
            filter(lambda p: p.requires_grad, model.parameters()),
            lr=lr, momentum=0.9
        )

        self.scheduler = lr_scheduler.StepLR(self.optimizer, step_size=7, gamma=0.1)

    def train(self):
        since = time.time()

        best_acc = 0
        best_model = copy.deepcopy(self.model.state_dict())

        for epoch in range(self.epochs):
            print(f"\nEpoch {epoch+1}/{self.epochs}")

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

                running_corrects = 0

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

                    self.optimizer.zero_grad()

                    with torch.set_grad_enabled(phase == 'train'):
                        outputs = self.model(inputs)
                        _, preds = torch.max(outputs, 1)
                        loss = self.criterion(outputs, labels)

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

                    running_corrects += torch.sum(preds == labels)

                if phase == 'train':
                    self.scheduler.step()

                epoch_acc = running_corrects.double() / self.dataset_sizes[phase]
                print(f"{phase} Acc: {epoch_acc:.4f}")

                if phase == 'val' and epoch_acc > best_acc:
                    best_acc = epoch_acc
                    best_model = copy.deepcopy(self.model.state_dict())

        time_elapsed = time.time() - since

        print(f"\nTraining complete in {int(time_elapsed//60)}m {int(time_elapsed%60)}s")
        print(f"Best val Acc: {best_acc:.6f}")
```

```
        self.model.load_state_dict(best_model)
        return self.model
```

```
def visualize_predictions(model, dataloaders, class_names, num_images=6):
    model.eval()
    images_so_far = 0
    fig = plt.figure(figsize=(10, 6))

    with torch.no_grad():
        for inputs, labels in 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"Pred: {class_names[preds[j]]}")

                img = inputs.cpu().data[j].permute(1, 2, 0)
                mean = torch.tensor([0.485, 0.456, 0.406])
                std = torch.tensor([0.229, 0.224, 0.225])
                img = std * img + mean
                img = torch.clamp(img, 0, 1)

                plt.imshow(img)

                if images_so_far == num_images:
                    return
```

Finetuning

```
data = DataModule("/content/drive/MyDrive/3YS2/CPE 313/catdog")
```

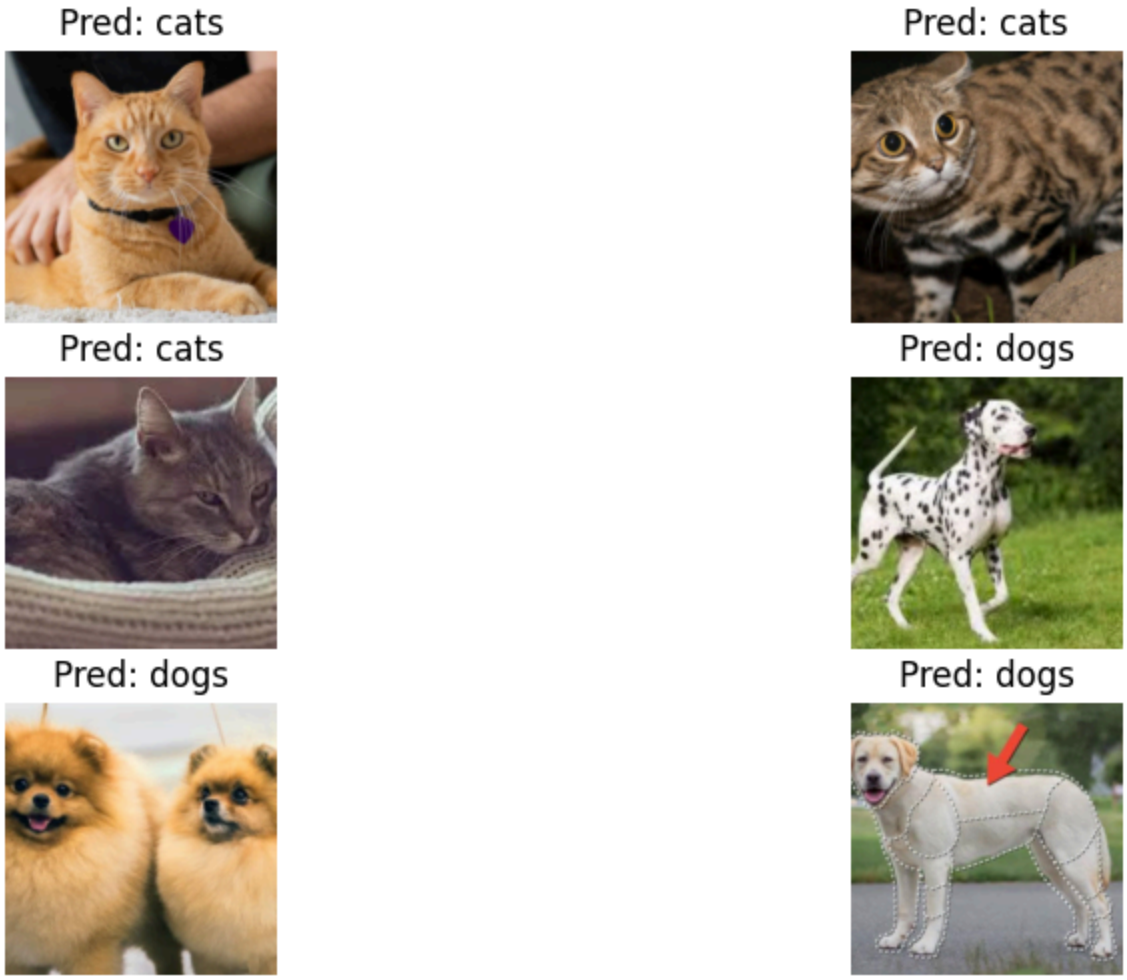
```
/usr/local/lib/python3.12/dist-packages/torch/utils/data/dataloader.py:424: User
  self.check_worker_number_rationality()
```

```
model_ft = ModelMaker(
    num_classes=len(data.class_names),
    feature_extract=False,
    model_name="efficientnet_v2_s"
).create()

trainer = Trainer(model_ft, data.dataloaders, data.dataset_sizes)
model_ft = trainer.train()
```

```
Downloading: "https://download.pytorch.org/models/efficientnet_v2_s-dd5fe13b.p
100%|██████████| 82.7M/82.7M [00:00<00:00, 213MB/s]
```

```
Epoch 1/25
/usr/local/lib/python3.12/dist-packages/torch/utils/data/dataloader.py:432: Us
  self.check_worker_number_rationality()
train Acc: 0.6830
val Acc: 0.9571

Epoch 2/25
train Acc: 0.8914
val Acc: 0.9714

Epoch 3/25
train Acc: 0.9089
val Acc: 0.9714

Epoch 4/25
train Acc: 0.9247
val Acc: 0.9714

Epoch 5/25
train Acc: 0.9387
val Acc: 0.9714

Epoch 6/25
train Acc: 0.9475
val Acc: 0.9714

Epoch 7/25
train Acc: 0.9387
val Acc: 0.9786

Epoch 8/25
train Acc: 0.9457
val Acc: 0.9714

Epoch 9/25
train Acc: 0.9405
val Acc: 0.9643

Epoch 10/25
train Acc: 0.9405
val Acc: 0.9643

Epoch 11/25
train Acc: 0.9492
val Acc: 0.9714

Epoch 12/25
train Acc: 0.9475
val Acc: 0.9714

Epoch 13/25
train Acc: 0.9387
val Acc: 0.9714
```

```
visualize_predictions(model_ft, data.dataloaders, data.class_names, num_images=6)
```



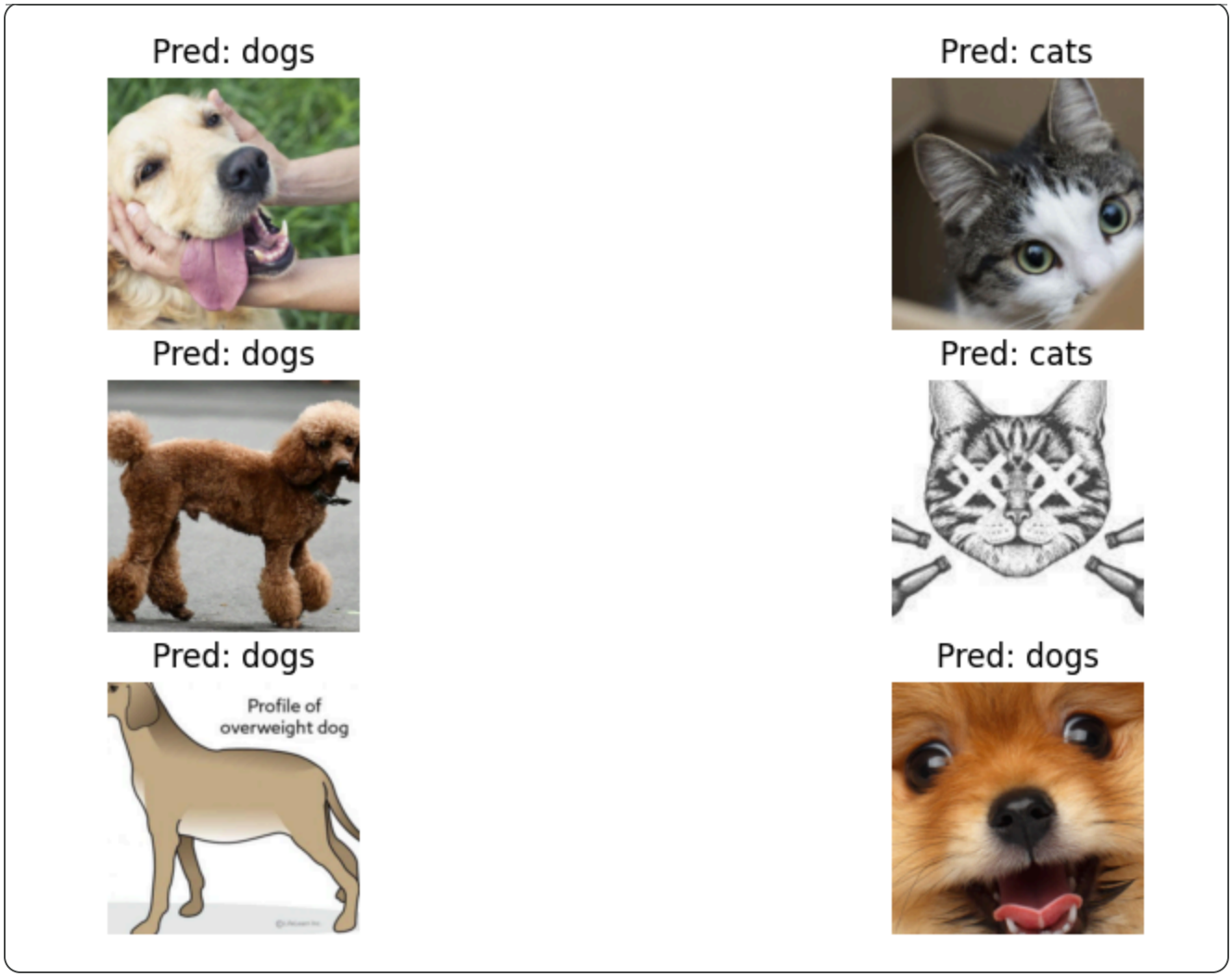
Feature Extraction

```
model_fe = ModelMaker(
    num_classes=len(data.class_names),
    feature_extract=True,
    model_name="efficientnet_v2_s"
).create()

trainer = Trainer(model_fe, data.dataloaders, data.dataset_sizes)
model_fe = trainer.train()
```

```
Epoch 1/25
train Acc: 0.6497
val Acc: 0.9214

Epoch 2/25
train Acc: 0.8441
val Acc: 0.9500

Epoch 3/25
```

```
train Acc: 0.8967
val Acc: 0.9571

Epoch 4/25
train Acc: 0.9089
val Acc: 0.9429

Epoch 5/25
train Acc: 0.9142
val Acc: 0.9714

Epoch 6/25
train Acc: 0.9142
val Acc: 0.9429

Epoch 7/25
train Acc: 0.9335
val Acc: 0.9429

Epoch 8/25
train Acc: 0.9159
val Acc: 0.9500

Epoch 9/25
train Acc: 0.9089
val Acc: 0.9500

Epoch 10/25
train Acc: 0.9370
val Acc: 0.9429

Epoch 11/25
train Acc: 0.9124
val Acc: 0.9500

Epoch 12/25
train Acc: 0.9159
val Acc: 0.9500

Epoch 13/25
train Acc: 0.9072
val Acc: 0.9643

Epoch 14/25
train Acc: 0.9282
val Acc: 0.9643

Epoch 15/25
```

```
visualize_predictions(model_fe, data.dataloaders, data.class_names, num_images=6)
```



Note(s) and Conclusion

Both Efficient-Net V2 models achieved validation accuracies above 95% for the cat-dog prediction/classification. The feature extraction approach trained almost twice as fast (3m 51s) as the finetuning approach (6m 44s) while producing a comparable accuracy (97.14% vs. 97.86%). This shows that finetuning the entire pretrained network yields slightly stronger predictive performance because the learned feature representations are allowed to adapt to the target dataset. In contrast, training only the classifier significantly reduces training time while still maintaining competitive accuracy. Also, I converted the original code into a class-based structure so it can be reused as a guide for future image datasets. This makes the workflow more organized and easier to adapt without rewriting the entire pipeline. I also used Google Colab's built-in AI to help detect errors and suggest more efficient code. Through this, the

> implementation became more scalable and practical for repeated experiments.