

Prac Deep Learning Sys (COMS 6998-015)

Homework 3

Nathan Coulibaly, nmc2169

Problem 1

1. Evaluation of pretrained MobilenetV1 SSD (mobilenet-v1-ssd-mp-0_675.pth) with Pascal VOC 2007 Dataset:

Average Precision Per Class:

- **aeroplane:** 0.6843
- **bicycle:** 0.7911
- **bird:** 0.6172
- **boat:** 0.5613
- **bottle:** 0.3483
- **bus:** 0.7684
- **car:** 0.7281
- **cat:** 0.8369
- **chair:** 0.5169
- **cow:** 0.6239
- **diningtable:** 0.7063
- **dog:** 0.7873
- **horse:** 0.8195
- **motorbike:** 0.7924
- **person:** 0.7023
- **pottedplant:** 0.3985
- **sheep:** 0.6067
- **sofa:** 0.7572
- **train:** 0.8262
- **tvmonitor:** 0.6465

Average Precision Across All Classes: 0.6760

Pretrained mobilenet-v1-ssd-mp-0_675 model with 3 classes (BACKGROUND, Airplane, Helicopter)

Average Precision Per-class:

- **Airplane:** 0.007641559442716817
- **Helicopter:** 0.005506622493994895

Average Precision Across All Classes: 0.006574090968355857

After fine-tuning for 10 epochs on open_images airplane and helicopter dataset, Validation Loss: 1.9261, Validation Regression Loss 0.6156, Validation Classification Loss: 1.3105

Average Precision Per-class:

- **Airplane:** 0.7871694636324705
- **Helicopter:** 0.8729167315760018

Average Precision Across All Classes: 0.8300430976042361

```

In [1]: import torch
import torch.onnx
from vision.ssd.mobilenetv1_ssd import create_mobilenetv1_ssd # Adjust the import if necessary

model_path = "models/mb1-ssd-Epoch-9-Loss-1.9260564812010141.pth"
model = create_mobilenetv1_ssd(num_classes=3)
model.load_state_dict(torch.load(model_path))
model.eval() # Set the model to evaluation mode

# Define a dummy input tensor
dummy_input = torch.randn(1, 3, 300, 300)

with torch.no_grad():
    pytorch_output = model(dummy_input)

print(f"PyTorch output: {pytorch_output}")

# Export the model to ONNX format
onnx_path = "models/finetuned-mb1-ssd.onnx"
torch.onnx.export(
    model, # Model to be exported
    dummy_input, # Dummy input tensor
    onnx_path, # Output file path
    export_params=True, # Store the trained parameter weights inside the model file
    opset_version=11, # ONNX opset version to export the model
    do_constant_folding=True, # Whether to execute constant folding for optimization
    input_names=['input'], # Input name (optional)
    output_names=['output'] # Output name (optional)
)

print(f"Model exported to {onnx_path}")

```

/tmp/ipykernel_31747/425630200.py:7: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default value), which uses the default pickle module implicitly. It is possible to construct malicious pickle data which will execute arbitrary code during unpickling (See <https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models> for more details). In a future release, the default value for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpickling. Arbitrary objects will no longer be allowed to be loaded via this mode unless they are explicitly allowlisted by the user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only=True` for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any issues related to this experimental feature.

```
model.load_state_dict(torch.load(model_path))
```

PyTorch output: (tensor([[[2.7067, -0.0481, -2.5708],
[3.0105, -0.3939, -2.5286],
[2.6784, -0.0125, -2.6966],
...,
[2.1973, -0.0763, -2.1296],
[2.4602, 0.1056, -2.5215],
[2.1238, -0.1471, -2.0171]]]), tensor([[[1.2836, 1.5608, -
4.3793, -5.8038],
[0.1748, 0.3736, -0.5482, -1.3703],
[1.3982, 0.6396, -5.4957, -4.8404],
...,
[0.0265, 0.1725, 1.6046, -0.3113],
[0.1325, 0.2001, -0.3001, 2.1285],
[-0.0083, 0.1599, 2.5981, -0.2166]]]))
Model exported to models/finetuned-mb1-ssd.onnx

```
In [2]: import onnx

# Load the ONNX model
onnx_model_path = "models/finetuned-mb1-ssd.onnx"
model = onnx.load(onnx_model_path)

# Verify the model's structure
try:
    onnx.checker.check_model(model)
    print("The ONNX model is valid.")
except onnx.checker.ValidationError as e:
    print(f"The ONNX model is invalid: {e}")
```

The ONNX model is valid.

```
In [3]: import onnxruntime as ort
import numpy as np

# Load the ONNX model
onnx_model_path = "models/finetuned-mb1-ssd.onnx"
ort_session = ort.InferenceSession(onnx_model_path)

# Define a dummy input tensor
dummy_input = np.random.randn(1, 3, 300, 300).astype(np.float32)

# Run the ONNX model with the dummy input tensor
input_name = ort_session.get_inputs()[0].name
output_name = ort_session.get_outputs()[0].name

# Run the ONNX inference
outputs = ort_session.run([output_name], {input_name: dummy_input})

print("Model Output:", outputs[0])
```

```
Model Output: [[[ 2.628398   -0.09165876 -2.416489   ]
 [ 3.0297675  -0.4312416  -2.5319319   ]
 [ 2.660162    0.05083796 -2.7098558   ]
 ...
 [ 2.2011666  -0.07641503 -2.132676   ]
 [ 2.469308    0.10345399 -2.5300725   ]
 [ 2.128943   -0.14993942 -2.0211074   ]]]
```

```
In [4]: pytorch_output_np = pytorch_output[0].detach().cpu().numpy() # Adjust
indexing if model returns multiple outputs
onnx_output = outputs[0]

if np.allclose(pytorch_output_np, onnx_output, rtol=1e-5, atol=1e-5):
    print("The outputs from PyTorch and ONNX Runtime match within the
specified tolerance.")
else:
    print("The outputs from PyTorch and ONNX Runtime do not match.")
```

The outputs from PyTorch and ONNX Runtime do not match.

- `rtol=1e-5`: Relative tolerance of $1e-5$ means that the difference between PyTorch and ONNX values can be up to 0.001% of the PyTorch value and still be considered a match.
- `atol=1e-5`: Absolute tolerance of $1e-5$ means that very small values (close to zero) can differ by up to $1e-5$.

```
In [5]: from PIL import Image, ImageDraw, ImageFont
import torchvision.transforms as transforms
```

```

In [6]: # Define paths to the images and model
image_paths = ["data/test_images/airplane.jpg", "data/test_images/heli
copter.jpg"]
onnx_model_path = "models/finetuned-mb1-ssd.onnx"

# Create an inference session with ONNX Runtime
session = ort.InferenceSession(onnx_model_path)

# Preprocessing function
def preprocess_image(image_path, input_size=(300, 300)):
    # Load the image
    image = Image.open(image_path).convert("RGB")

    # Define preprocessing steps (resize, normalize, etc.)
    preprocess = transforms.Compose([
        transforms.Resize(input_size),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.
5]) # Adjust mean/std based on model needs
    ])

    # Apply transformations
    image_tensor = preprocess(image)

    # Add batch dimension
    image_tensor = image_tensor.unsqueeze(0) # Shape: (1, 3, 300, 30
0)

    return image_tensor.numpy() # Convert to numpy for ONNX Runtime

```

```
In [7]: for image_path in image_paths:
        # Preprocess the image
        input_data = preprocess_image(image_path)

        # Get input and output names for ONNX Runtime
        input_name = session.get_inputs()[0].name
        output_name = session.get_outputs()[0].name

        # Run inference
        outputs = session.run([output_name], {input_name: input_data})

        # Print the output (for real applications, interpret this output properly)
        print(f"Output for {image_path}: {outputs[0]}")
```

```
Output for data/test_images/airplane.jpg: [[[ 4.138382  -0.9878388  -
3.0577788 ]
 [ 4.22545   -1.4909672  -2.6169086 ]
 [ 3.8684378  -1.0116055  -2.864805   ]
 ...
 [ 0.5566062   1.7188514  -2.3376985 ]
 [-1.7088915   3.212818   -1.5116981 ]
 [ 2.3896177   0.58438176 -2.919279   ]]]
Output for data/test_images/helicopter.jpg: [[[ 4.265738  -0.82885605
-3.3536644 ]
 [ 4.3293333  -0.9208403  -3.3959553 ]
 [ 4.0182996  -0.63204706 -3.3168209 ]
 ...
 [ 0.50133926 -2.3493757   1.7498918 ]
 [-0.7328597  -1.7276728   2.302973   ]
 [ 1.6628044  -2.7264628   1.0709631 ]]]
```

```
In [33]: labels = ["airplane", "helicopter", "BACKGROUND"]
```

```
In [34]: def preprocess_image(image_path, input_size=(300, 300)):
        image = Image.open(image_path).convert("RGB")
        preprocess = transforms.Compose([
            transforms.Resize(input_size),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.
5]) # Adjust mean/std as needed
        ])
        image_tensor = preprocess(image)
        image_tensor = image_tensor.unsqueeze(0)
        return image_tensor.numpy(), image
```

```
In [35]: def parse_output(outputs, threshold=0.5):
# Extract the single tensor from outputs
detections = outputs[0]

# Initialize lists for results
result_boxes = [] # If there are no bounding boxes, this remains
empty
result_labels = []
result_scores = []

# Iterate over the detections
for detection in detections[0]: # Iterate over 3000 detections
    class_confidence, class_id, _ = detection # Hypothetical structure
    if class_confidence > threshold:
        result_labels.append(labels[int(class_id)]) # Assuming class_id corresponds to a label index
        result_scores.append(class_confidence)
        # Add a dummy box if your SSD model is supposed to provide bounding boxes
        result_boxes.append([0, 0, 1, 1]) # Dummy bounding box (replace with actual if available)

    return result_boxes, result_labels, result_scores
```


In [36]: `import matplotlib.pyplot as plt`

```
for image_path in image_paths:
    # Preprocess the image
    input_data, original_image = preprocess_image(image_path)

    # Run inference
    input_name = session.get_inputs()[0].name
    output_name = session.get_outputs()[0].name
    outputs = session.run([output_name], {input_name: input_data})

    # Parse the output
    result_boxes, result_labels, result_scores = parse_output(outputs)

    # Annotate the image
    draw = ImageDraw.Draw(original_image)
    font = ImageFont.load_default()
    for box, label, score in zip(result_boxes, result_labels, result_scores):
        xmin, ymin, xmax, ymax = box
        xmin, ymin, xmax, ymax = int(xmin * original_image.width), int
        (ymin * original_image.height), int(xmax * original_image.width), int
        (ymax * original_image.height)

        # Draw bounding box
        draw.rectangle([(xmin, ymin), (xmax, ymax)], outline="red", width=2)

        # Draw label and score
        text = f"{label}: {score:.2f}"
        text_size = font.getbbox(text) # Get the bounding box of the
        text

        text_width = text_size[2] - text_size[0]
        text_height = text_size[3] - text_size[1]

        # Draw the background rectangle for the label
        draw.rectangle([(xmin, ymin - text_height), (xmin + text_width,
        ymin)], fill="red")
        draw.text((xmin, ymin - text_height), text, fill="white", font=font)

    # Display the annotated image
    plt.figure(figsize=(8, 8))
    plt.imshow(original_image)
    plt.axis("off")
    plt.title(f"Inference Result for {label}")
    plt.show()
```

Inference Result for airplane



Inference Result for helicopter



Problem 2

1. Fine-tuning with Daimler Ped dataset from the Visual Domain Decathlon

a.

```
In [45]: import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim.lr_scheduler import MultiStepLR
from torchvision import models, transforms
from torch.utils.data import DataLoader
from torchvision.datasets import ImageFolder
```

```
In [46]: data_transforms = {
    'train': transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.2
25])
    ]),
    'val': transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.2
25])
    ]),
}
```

```
In [47]: data_dir = 'data/daimlerpedcls'
image_datasets = {
    'train': ImageFolder(data_dir + '/train', data_transforms['train']),
    'val': ImageFolder(data_dir + '/val', data_transforms['val']),
}
dataloaders = {
    'train': DataLoader(image_datasets['train'], batch_size=64, shuffle=True, num_workers=4),
    'val': DataLoader(image_datasets['val'], batch_size=64, shuffle=False, num_workers=4),
}
dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'val']}
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

b.

```
In [48]: model_ft = models.resnet50(pretrained=True)
num_ftrs = model_ft.fc.in_features
model_ft.fc = nn.Linear(num_ftrs, 2) # 2 classes for the Daimler Ped dataset
model_ft = model_ft.to(device)
```

```
In [49]: criterion = nn.CrossEntropyLoss()
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)

scheduler = MultiStepLR(optimizer_ft, milestones=[40, 80, 120], gamma=0.1)
```

```

In [ ]: num_epochs = 100
        for epoch in range(num_epochs):
            print(f'Epoch {epoch}/{num_epochs - 1}')
            print('-' * 10)

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

            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_ft.zero_grad()

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

                # Backward + optimize only if in training phase
                if phase == 'train':
                    loss.backward()
                    optimizer_ft.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}')

        print('Training complete')

```

Epoch 0/99

train Loss: 0.0953 Acc: 0.9618
val Loss: 0.0175 Acc: 0.9942
Epoch 1/99

train Loss: 0.0122 Acc: 0.9964
val Loss: 0.0078 Acc: 0.9973
Epoch 2/99

train Loss: 0.0057 Acc: 0.9985
val Loss: 0.0051 Acc: 0.9974
Epoch 3/99

train Loss: 0.0038 Acc: 0.9989
val Loss: 0.0033 Acc: 0.9991
Epoch 4/99

train Loss: 0.0021 Acc: 0.9994
val Loss: 0.0037 Acc: 0.9988
Epoch 5/99

train Loss: 0.0018 Acc: 0.9996
val Loss: 0.0034 Acc: 0.9985
Epoch 6/99

train Loss: 0.0020 Acc: 0.9995
val Loss: 0.0024 Acc: 0.9993
Epoch 7/99

train Loss: 0.0011 Acc: 0.9998
val Loss: 0.0021 Acc: 0.9988
Epoch 8/99

train Loss: 0.0006 Acc: 1.0000
val Loss: 0.0017 Acc: 0.9995
Epoch 9/99

train Loss: 0.0006 Acc: 1.0000
val Loss: 0.0017 Acc: 0.9991
Epoch 10/99

train Loss: 0.0007 Acc: 1.0000
val Loss: 0.0014 Acc: 0.9995
Epoch 11/99

train Loss: 0.0005 Acc: 0.9999
val Loss: 0.0014 Acc: 0.9995
Epoch 12/99

train Loss: 0.0008 Acc: 0.9998
val Loss: 0.0022 Acc: 0.9993
Epoch 13/99

train Loss: 0.0004 Acc: 1.0000
val Loss: 0.0016 Acc: 0.9991
Epoch 14/99

train Loss: 0.0005 Acc: 0.9999
val Loss: 0.0017 Acc: 0.9993
Epoch 15/99

train Loss: 0.0005 Acc: 0.9999
val Loss: 0.0017 Acc: 0.9993
Epoch 16/99

train Loss: 0.0004 Acc: 0.9999
val Loss: 0.0014 Acc: 0.9993
Epoch 17/99

train Loss: 0.0003 Acc: 1.0000
val Loss: 0.0013 Acc: 0.9993
Epoch 18/99

train Loss: 0.0004 Acc: 0.9999
val Loss: 0.0018 Acc: 0.9995
Epoch 19/99

train Loss: 0.0002 Acc: 1.0000
val Loss: 0.0018 Acc: 0.9995
Epoch 20/99

train Loss: 0.0002 Acc: 1.0000
val Loss: 0.0015 Acc: 0.9997
Epoch 21/99

train Loss: 0.0003 Acc: 1.0000
val Loss: 0.0019 Acc: 0.9991
Epoch 22/99

train Loss: 0.0002 Acc: 0.9999
val Loss: 0.0015 Acc: 0.9995
Epoch 23/99

train Loss: 0.0001 Acc: 1.0000
val Loss: 0.0014 Acc: 0.9995
Epoch 24/99

train Loss: 0.0002 Acc: 1.0000
val Loss: 0.0014 Acc: 0.9993
Epoch 25/99

train Loss: 0.0002 Acc: 1.0000
val Loss: 0.0018 Acc: 0.9993
Epoch 26/99

train Loss: 0.0001 Acc: 1.0000
val Loss: 0.0015 Acc: 0.9993
Epoch 27/99

train Loss: 0.0002 Acc: 1.0000
val Loss: 0.0016 Acc: 0.9995
Epoch 28/99

```
train Loss: 0.0002 Acc: 1.0000
val Loss: 0.0015 Acc: 0.9995
Epoch 29/99
-----
train Loss: 0.0002 Acc: 1.0000
val Loss: 0.0016 Acc: 0.9993
Epoch 30/99
-----
train Loss: 0.0001 Acc: 1.0000
val Loss: 0.0015 Acc: 0.9993
Epoch 31/99
-----
train Loss: 0.0001 Acc: 1.0000
val Loss: 0.0015 Acc: 0.9993
Epoch 32/99
-----
train Loss: 0.0002 Acc: 1.0000
val Loss: 0.0015 Acc: 0.9993
Epoch 33/99
-----
train Loss: 0.0002 Acc: 1.0000
val Loss: 0.0016 Acc: 0.9993
Epoch 34/99
-----
train Loss: 0.0002 Acc: 1.0000
val Loss: 0.0015 Acc: 0.9995
Epoch 35/99
-----
train Loss: 0.0001 Acc: 1.0000
val Loss: 0.0018 Acc: 0.9993
Epoch 36/99
-----
train Loss: 0.0002 Acc: 1.0000
val Loss: 0.0014 Acc: 0.9995
Epoch 37/99
-----
train Loss: 0.0003 Acc: 1.0000
val Loss: 0.0017 Acc: 0.9995
Epoch 38/99
-----
train Loss: 0.0001 Acc: 1.0000
val Loss: 0.0015 Acc: 0.9995
Epoch 39/99
-----
train Loss: 0.0002 Acc: 1.0000
val Loss: 0.0014 Acc: 0.9995
Epoch 40/99
-----
train Loss: 0.0002 Acc: 1.0000
val Loss: 0.0015 Acc: 0.9993
Epoch 41/99
-----
train Loss: 0.0001 Acc: 1.0000
val Loss: 0.0014 Acc: 0.9995
Epoch 42/99
-----
train Loss: 0.0001 Acc: 1.0000
```



```
val Loss: 0.0014 Acc: 0.9993
Epoch 43/99
-----
train Loss: 0.0002 Acc: 1.0000
val Loss: 0.0015 Acc: 0.9995
Epoch 44/99
-----
train Loss: 0.0001 Acc: 1.0000
val Loss: 0.0022 Acc: 0.9993
Epoch 45/99
-----
train Loss: 0.0001 Acc: 1.0000
val Loss: 0.0015 Acc: 0.9995
Epoch 46/99
-----
train Loss: 0.0001 Acc: 1.0000
val Loss: 0.0013 Acc: 0.9997
Epoch 47/99
-----
train Loss: 0.0002 Acc: 1.0000
val Loss: 0.0015 Acc: 0.9995
Epoch 48/99
-----
train Loss: 0.0001 Acc: 1.0000
val Loss: 0.0013 Acc: 0.9995
Epoch 49/99
-----
train Loss: 0.0001 Acc: 1.0000
val Loss: 0.0015 Acc: 0.9993
Epoch 50/99
-----
train Loss: 0.0001 Acc: 1.0000
val Loss: 0.0016 Acc: 0.9993
Epoch 51/99
-----
train Loss: 0.0002 Acc: 1.0000
val Loss: 0.0015 Acc: 0.9995
Epoch 52/99
-----
train Loss: 0.0002 Acc: 1.0000
val Loss: 0.0016 Acc: 0.9993
Epoch 53/99
-----
train Loss: 0.0001 Acc: 1.0000
val Loss: 0.0015 Acc: 0.9995
Epoch 54/99
-----
train Loss: 0.0003 Acc: 0.9999
val Loss: 0.0014 Acc: 0.9997
Epoch 55/99
-----
train Loss: 0.0002 Acc: 1.0000
val Loss: 0.0019 Acc: 0.9993
Epoch 56/99
-----
train Loss: 0.0001 Acc: 1.0000
val Loss: 0.0015 Acc: 0.9993
```

Epoch 57/99

train Loss: 0.0001 Acc: 1.0000

val Loss: 0.0014 Acc: 0.9993

Epoch 58/99

train Loss: 0.0002 Acc: 1.0000

val Loss: 0.0014 Acc: 0.9995

Epoch 59/99

train Loss: 0.0001 Acc: 1.0000

val Loss: 0.0017 Acc: 0.9991

Epoch 60/99

train Loss: 0.0001 Acc: 1.0000

val Loss: 0.0014 Acc: 0.9995

Epoch 61/99

train Loss: 0.0001 Acc: 1.0000

val Loss: 0.0014 Acc: 0.9995

Epoch 62/99

train Loss: 0.0001 Acc: 1.0000

val Loss: 0.0016 Acc: 0.9993

Epoch 63/99

train Loss: 0.0001 Acc: 1.0000

val Loss: 0.0017 Acc: 0.9993

Epoch 64/99

train Loss: 0.0003 Acc: 0.9999

val Loss: 0.0013 Acc: 0.9997

Epoch 65/99

train Loss: 0.0001 Acc: 1.0000

val Loss: 0.0014 Acc: 0.9997

Epoch 66/99

train Loss: 0.0001 Acc: 1.0000

val Loss: 0.0015 Acc: 0.9995

Epoch 67/99

train Loss: 0.0001 Acc: 1.0000

val Loss: 0.0014 Acc: 0.9995

Epoch 68/99

train Loss: 0.0001 Acc: 1.0000

val Loss: 0.0014 Acc: 0.9995

Epoch 69/99

train Loss: 0.0001 Acc: 1.0000

val Loss: 0.0014 Acc: 0.9995

Epoch 70/99

train Loss: 0.0002 Acc: 1.0000

val Loss: 0.0013 Acc: 0.9993

Epoch 71/99

train Loss: 0.0002 Acc: 1.0000
val Loss: 0.0014 Acc: 0.9995
Epoch 72/99

train Loss: 0.0001 Acc: 1.0000
val Loss: 0.0016 Acc: 0.9993
Epoch 73/99

train Loss: 0.0002 Acc: 1.0000
val Loss: 0.0015 Acc: 0.9993
Epoch 74/99

train Loss: 0.0001 Acc: 1.0000
val Loss: 0.0017 Acc: 0.9997
Epoch 75/99

train Loss: 0.0001 Acc: 1.0000
val Loss: 0.0014 Acc: 0.9995
Epoch 76/99

train Loss: 0.0001 Acc: 1.0000
val Loss: 0.0017 Acc: 0.9993
Epoch 77/99

train Loss: 0.0001 Acc: 1.0000
val Loss: 0.0014 Acc: 0.9995
Epoch 78/99

train Loss: 0.0002 Acc: 1.0000
val Loss: 0.0016 Acc: 0.9993
Epoch 79/99

train Loss: 0.0001 Acc: 1.0000
val Loss: 0.0015 Acc: 0.9995
Epoch 80/99

train Loss: 0.0002 Acc: 1.0000
val Loss: 0.0015 Acc: 0.9993
Epoch 81/99

train Loss: 0.0002 Acc: 1.0000
val Loss: 0.0014 Acc: 0.9997
Epoch 82/99

train Loss: 0.0002 Acc: 1.0000
val Loss: 0.0013 Acc: 0.9995
Epoch 83/99

train Loss: 0.0001 Acc: 1.0000
val Loss: 0.0019 Acc: 0.9993
Epoch 84/99

train Loss: 0.0001 Acc: 1.0000
val Loss: 0.0014 Acc: 0.9993
Epoch 85/99

```
train Loss: 0.0001 Acc: 1.0000
val Loss: 0.0016 Acc: 0.9995
Epoch 86/99
-----
train Loss: 0.0001 Acc: 1.0000
val Loss: 0.0015 Acc: 0.9993
Epoch 87/99
-----
train Loss: 0.0001 Acc: 1.0000
val Loss: 0.0014 Acc: 0.9995
Epoch 88/99
-----
train Loss: 0.0001 Acc: 1.0000
val Loss: 0.0016 Acc: 0.9995
Epoch 89/99
-----
train Loss: 0.0002 Acc: 1.0000
val Loss: 0.0014 Acc: 0.9995
Epoch 90/99
-----
train Loss: 0.0002 Acc: 0.9999
val Loss: 0.0013 Acc: 0.9997
Epoch 91/99
-----
train Loss: 0.0001 Acc: 1.0000
val Loss: 0.0015 Acc: 0.9995
Epoch 92/99
-----
train Loss: 0.0002 Acc: 1.0000
val Loss: 0.0015 Acc: 0.9995
Epoch 93/99
-----
train Loss: 0.0001 Acc: 1.0000
val Loss: 0.0017 Acc: 0.9993
Epoch 94/99
-----
train Loss: 0.0001 Acc: 1.0000
val Loss: 0.0016 Acc: 0.9995
Epoch 95/99
-----
train Loss: 0.0002 Acc: 1.0000
val Loss: 0.0014 Acc: 0.9995
Epoch 96/99
-----
train Loss: 0.0002 Acc: 1.0000
val Loss: 0.0015 Acc: 0.9995
Epoch 97/99
-----
train Loss: 0.0001 Acc: 1.0000
val Loss: 0.0014 Acc: 0.9993
Epoch 98/99
-----
train Loss: 0.0001 Acc: 1.0000
val Loss: 0.0013 Acc: 0.9997
Epoch 99/99
-----
train Loss: 0.0001 Acc: 1.0000
```

val Loss: 0.0014 Acc: 0.9995
Training complete

c.

```

In [ ]: import copy

def train_model(model, criterion, optimizer, scheduler, num_epochs=100):
    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)

        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)
                    _, preds = torch.max(outputs, 1)
                    loss = criterion(outputs, labels)

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

                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}')

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

        print('Training complete')
        print(f'Best val Acc: {best_acc:.4f}')
        model.load_state_dict(best_model_wts)
        return model, best_acc

```

```
In [ ]: # Experiment 1: Learning rate = 0.001 (already done, refer to previous results)
# Experiment 2: Learning rate = 0.01
model_ft_01 = models.resnet50(pretrained=True)
model_ft_01.fc = nn.Linear(model_ft_01.fc.in_features, 2)
model_ft_01 = model_ft_01.to(device)
optimizer_01 = optim.SGD(model_ft_01.parameters(), lr=0.01, momentum=0.9)
scheduler_01 = MultiStepLR(optimizer_01, milestones=[25, 50, 75], gamma=0.1)
model_ft_01, best_acc_01 = train_model(model_ft_01, criterion, optimizer_01, scheduler_01)

# Experiment 3: Learning rate = 0.1
model_ft_1 = models.resnet50(pretrained=True)
model_ft_1.fc = nn.Linear(model_ft_1.fc.in_features, 2)
model_ft_1 = model_ft_1.to(device)
optimizer_1 = optim.SGD(model_ft_1.parameters(), lr=0.1, momentum=0.9)
scheduler_1 = MultiStepLR(optimizer_1, milestones=[25, 50, 75], gamma=0.1)
model_ft_1, best_acc_1 = train_model(model_ft_1, criterion, optimizer_1, scheduler_1)
```

```
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:2
08: UserWarning: The parameter 'pretrained' is deprecated since 0.13 a
nd may be removed in the future, please use 'weights' instead.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:2
23: UserWarning: Arguments other than a weight enum or `None` for 'wei
ghts' are deprecated since 0.13 and may be removed in the future. The
current behavior is equivalent to passing `weights=ResNet50_Weights.IM
AGENET1K_V1`. You can also use `weights=ResNet50_Weights.DEFAULT` to g
et the most up-to-date weights.
  warnings.warn(msg)
```



```
Epoch 0/99
-----
train Loss: 0.0573 Acc: 0.9767
val Loss: 0.0069 Acc: 0.9973
Epoch 1/99
-----
train Loss: 0.0055 Acc: 0.9981
val Loss: 0.0022 Acc: 0.9993
Epoch 2/99
-----
train Loss: 0.0022 Acc: 0.9991
val Loss: 0.0027 Acc: 0.9993
Epoch 3/99
-----
train Loss: 0.0015 Acc: 0.9996
val Loss: 0.0015 Acc: 0.9995
Epoch 4/99
-----
train Loss: 0.0006 Acc: 0.9998
val Loss: 0.0011 Acc: 0.9997
Epoch 5/99
-----
train Loss: 0.0002 Acc: 0.9999
val Loss: 0.0014 Acc: 0.9997
Epoch 6/99
-----
train Loss: 0.0005 Acc: 0.9999
val Loss: 0.0006 Acc: 0.9995
Epoch 7/99
-----
train Loss: 0.0005 Acc: 0.9999
val Loss: 0.0009 Acc: 0.9995
Epoch 8/99
-----
train Loss: 0.0001 Acc: 1.0000
val Loss: 0.0009 Acc: 0.9995
Epoch 9/99
-----
train Loss: 0.0001 Acc: 1.0000
val Loss: 0.0007 Acc: 0.9998
Epoch 10/99
-----
train Loss: 0.0001 Acc: 1.0000
val Loss: 0.0005 Acc: 0.9997
Epoch 11/99
-----
train Loss: 0.0001 Acc: 1.0000
val Loss: 0.0002 Acc: 1.0000
Epoch 12/99
-----
train Loss: 0.0001 Acc: 1.0000
val Loss: 0.0005 Acc: 0.9998
Epoch 13/99
-----
train Loss: 0.0001 Acc: 1.0000
val Loss: 0.0004 Acc: 0.9998
Epoch 14/99
```

train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0004 Acc: 0.9997
Epoch 15/99

train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0003 Acc: 0.9998
Epoch 16/99

train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0002 Acc: 1.0000
Epoch 17/99

train Loss: 0.0001 Acc: 1.0000
val Loss: 0.0005 Acc: 0.9997
Epoch 18/99

train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0004 Acc: 0.9997
Epoch 19/99

train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0008 Acc: 0.9997
Epoch 20/99

train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0003 Acc: 1.0000
Epoch 21/99

train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0002 Acc: 1.0000
Epoch 22/99

train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0002 Acc: 1.0000
Epoch 23/99

train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0001 Acc: 1.0000
Epoch 24/99

train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0002 Acc: 1.0000
Epoch 25/99

train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0001 Acc: 1.0000
Epoch 26/99

train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0002 Acc: 1.0000
Epoch 27/99

train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0003 Acc: 1.0000
Epoch 28/99

```
train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0002 Acc: 1.0000
Epoch 29/99
-----
train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0003 Acc: 0.9998
Epoch 30/99
-----
train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0002 Acc: 1.0000
Epoch 31/99
-----
train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0002 Acc: 1.0000
Epoch 32/99
-----
train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0003 Acc: 0.9998
Epoch 33/99
-----
train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0002 Acc: 1.0000
Epoch 34/99
-----
train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0002 Acc: 1.0000
Epoch 35/99
-----
train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0002 Acc: 1.0000
Epoch 36/99
-----
train Loss: 0.0002 Acc: 0.9999
val Loss: 0.0001 Acc: 1.0000
Epoch 37/99
-----
train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0002 Acc: 1.0000
Epoch 38/99
-----
train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0002 Acc: 1.0000
Epoch 39/99
-----
train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0001 Acc: 1.0000
Epoch 40/99
-----
train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0002 Acc: 1.0000
Epoch 41/99
-----
train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0004 Acc: 0.9998
Epoch 42/99
-----
train Loss: 0.0000 Acc: 1.0000
```

```
val Loss: 0.0003 Acc: 1.0000
Epoch 43/99
-----
train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0002 Acc: 1.0000
Epoch 44/99
-----
train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0002 Acc: 1.0000
Epoch 45/99
-----
train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0002 Acc: 1.0000
Epoch 46/99
-----
train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0002 Acc: 1.0000
Epoch 47/99
-----
train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0001 Acc: 1.0000
Epoch 48/99
-----
train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0002 Acc: 1.0000
Epoch 49/99
-----
train Loss: 0.0001 Acc: 1.0000
val Loss: 0.0009 Acc: 0.9995
Epoch 50/99
-----
train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0002 Acc: 1.0000
Epoch 51/99
-----
train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0003 Acc: 0.9998
Epoch 52/99
-----
train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0005 Acc: 0.9997
Epoch 53/99
-----
train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0004 Acc: 0.9997
Epoch 54/99
-----
train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0002 Acc: 1.0000
Epoch 55/99
-----
train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0001 Acc: 1.0000
Epoch 56/99
-----
train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0002 Acc: 1.0000
```

Epoch 57/99

train Loss: 0.0000 Acc: 1.0000

val Loss: 0.0003 Acc: 1.0000

Epoch 58/99

train Loss: 0.0000 Acc: 1.0000

val Loss: 0.0002 Acc: 1.0000

Epoch 59/99

train Loss: 0.0000 Acc: 1.0000

val Loss: 0.0001 Acc: 1.0000

Epoch 60/99

train Loss: 0.0000 Acc: 1.0000

val Loss: 0.0002 Acc: 1.0000

Epoch 61/99

train Loss: 0.0000 Acc: 1.0000

val Loss: 0.0003 Acc: 1.0000

Epoch 62/99

train Loss: 0.0000 Acc: 1.0000

val Loss: 0.0001 Acc: 1.0000

Epoch 63/99

train Loss: 0.0000 Acc: 1.0000

val Loss: 0.0002 Acc: 1.0000

Epoch 64/99

train Loss: 0.0000 Acc: 1.0000

val Loss: 0.0003 Acc: 1.0000

Epoch 65/99

train Loss: 0.0000 Acc: 1.0000

val Loss: 0.0003 Acc: 1.0000

Epoch 66/99

train Loss: 0.0000 Acc: 1.0000

val Loss: 0.0005 Acc: 0.9997

Epoch 67/99

train Loss: 0.0000 Acc: 1.0000

val Loss: 0.0003 Acc: 1.0000

Epoch 68/99

train Loss: 0.0000 Acc: 1.0000

val Loss: 0.0004 Acc: 0.9998

Epoch 69/99

train Loss: 0.0000 Acc: 1.0000

val Loss: 0.0004 Acc: 0.9997

Epoch 70/99

train Loss: 0.0000 Acc: 1.0000

val Loss: 0.0004 Acc: 0.9998

Epoch 71/99

train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0005 Acc: 0.9997
Epoch 72/99

train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0002 Acc: 1.0000
Epoch 73/99

train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0002 Acc: 1.0000
Epoch 74/99

train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0005 Acc: 0.9997
Epoch 75/99

train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0001 Acc: 1.0000
Epoch 76/99

train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0002 Acc: 1.0000
Epoch 77/99

train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0001 Acc: 1.0000
Epoch 78/99

train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0003 Acc: 0.9998
Epoch 79/99

train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0002 Acc: 1.0000
Epoch 80/99

train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0004 Acc: 0.9998
Epoch 81/99

train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0002 Acc: 1.0000
Epoch 82/99

train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0003 Acc: 1.0000
Epoch 83/99

train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0003 Acc: 0.9998
Epoch 84/99

train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0007 Acc: 0.9997
Epoch 85/99

```
train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0003 Acc: 1.0000
Epoch 86/99
-----
train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0002 Acc: 1.0000
Epoch 87/99
-----
train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0002 Acc: 1.0000
Epoch 88/99
-----
train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0003 Acc: 0.9998
Epoch 89/99
-----
train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0002 Acc: 1.0000
Epoch 90/99
-----
train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0005 Acc: 0.9997
Epoch 91/99
-----
train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0005 Acc: 0.9997
Epoch 92/99
-----
train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0002 Acc: 1.0000
Epoch 93/99
-----
train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0002 Acc: 1.0000
Epoch 94/99
-----
train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0002 Acc: 1.0000
Epoch 95/99
-----
train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0002 Acc: 1.0000
Epoch 96/99
-----
train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0002 Acc: 1.0000
Epoch 97/99
-----
train Loss: 0.0000 Acc: 1.0000
val Loss: 0.0002 Acc: 1.0000
Epoch 98/99
-----
train Loss: 0.0001 Acc: 1.0000
val Loss: 0.0002 Acc: 1.0000
Epoch 99/99
-----
train Loss: 0.0000 Acc: 1.0000
```

```
val Loss: 0.0003 Acc: 0.9998
Training complete
Best val Acc: 1.0000
Epoch 0/99
-----
train Loss: 0.9098 Acc: 0.6381
val Loss: 0.5428 Acc: 0.7332
Epoch 1/99
-----
train Loss: 0.4531 Acc: 0.7943
val Loss: 0.3540 Acc: 0.8420
Epoch 2/99
-----
train Loss: 0.3137 Acc: 0.8654
val Loss: 0.2986 Acc: 0.8827
Epoch 3/99
-----
train Loss: 0.2173 Acc: 0.9110
val Loss: 0.2294 Acc: 0.9063
Epoch 4/99
-----
train Loss: 0.1549 Acc: 0.9400
val Loss: 0.4398 Acc: 0.8417
Epoch 5/99
-----
train Loss: 0.1289 Acc: 0.9497
val Loss: 0.1286 Acc: 0.9549
Epoch 6/99
-----
train Loss: 0.1010 Acc: 0.9620
val Loss: 0.1408 Acc: 0.9466
Epoch 7/99
-----
train Loss: 0.0799 Acc: 0.9699
val Loss: 0.1289 Acc: 0.9531
Epoch 8/99
-----
train Loss: 0.0684 Acc: 0.9738
val Loss: 0.1812 Acc: 0.9260
Epoch 9/99
-----
train Loss: 0.0599 Acc: 0.9769
val Loss: 0.0651 Acc: 0.9743
Epoch 10/99
-----
train Loss: 0.0481 Acc: 0.9820
val Loss: 0.0698 Acc: 0.9757
Epoch 11/99
-----
train Loss: 0.0413 Acc: 0.9848
val Loss: 0.0321 Acc: 0.9878
Epoch 12/99
-----
train Loss: 0.0351 Acc: 0.9869
val Loss: 0.0617 Acc: 0.9774
Epoch 13/99
-----
```



```
train Loss: 0.0308 Acc: 0.9889
val Loss: 0.0585 Acc: 0.9782
Epoch 14/99
-----
train Loss: 0.0295 Acc: 0.9898
val Loss: 0.0322 Acc: 0.9889
Epoch 15/99
-----
train Loss: 0.0207 Acc: 0.9926
val Loss: 0.0399 Acc: 0.9855
Epoch 16/99
-----
train Loss: 0.0211 Acc: 0.9925
val Loss: 0.0311 Acc: 0.9901
Epoch 17/99
-----
train Loss: 0.0186 Acc: 0.9937
val Loss: 0.0285 Acc: 0.9891
Epoch 18/99
-----
train Loss: 0.0171 Acc: 0.9936
val Loss: 0.0257 Acc: 0.9896
Epoch 19/99
-----
train Loss: 0.0156 Acc: 0.9947
val Loss: 0.0284 Acc: 0.9903
Epoch 20/99
-----
train Loss: 0.0148 Acc: 0.9950
val Loss: 0.0544 Acc: 0.9815
Epoch 21/99
-----
train Loss: 0.0116 Acc: 0.9963
val Loss: 0.0139 Acc: 0.9946
Epoch 22/99
-----
train Loss: 0.0102 Acc: 0.9963
val Loss: 0.0291 Acc: 0.9915
Epoch 23/99
-----
train Loss: 0.0139 Acc: 0.9948
val Loss: 0.0466 Acc: 0.9867
Epoch 24/99
-----
train Loss: 0.0079 Acc: 0.9974
val Loss: 0.0138 Acc: 0.9956
Epoch 25/99
-----
train Loss: 0.0035 Acc: 0.9987
val Loss: 0.0095 Acc: 0.9961
Epoch 26/99
-----
train Loss: 0.0017 Acc: 0.9997
val Loss: 0.0083 Acc: 0.9969
Epoch 27/99
-----
train Loss: 0.0021 Acc: 0.9993
```

```
val Loss: 0.0066 Acc: 0.9978
Epoch 28/99
-----
train Loss: 0.0013 Acc: 0.9996
val Loss: 0.0066 Acc: 0.9983
Epoch 29/99
-----
train Loss: 0.0013 Acc: 0.9995
val Loss: 0.0074 Acc: 0.9976
Epoch 30/99
-----
train Loss: 0.0012 Acc: 0.9997
val Loss: 0.0075 Acc: 0.9980
Epoch 31/99
-----
train Loss: 0.0008 Acc: 0.9998
val Loss: 0.0068 Acc: 0.9980
Epoch 32/99
-----
train Loss: 0.0005 Acc: 0.9999
val Loss: 0.0094 Acc: 0.9971
Epoch 33/99
-----
train Loss: 0.0011 Acc: 0.9995
val Loss: 0.0070 Acc: 0.9978
Epoch 34/99
-----
train Loss: 0.0008 Acc: 0.9998
val Loss: 0.0073 Acc: 0.9976
Epoch 35/99
-----
train Loss: 0.0006 Acc: 0.9998
val Loss: 0.0078 Acc: 0.9978
Epoch 36/99
-----
train Loss: 0.0008 Acc: 0.9998
val Loss: 0.0061 Acc: 0.9981
Epoch 37/99
-----
train Loss: 0.0008 Acc: 0.9997
val Loss: 0.0074 Acc: 0.9973
Epoch 38/99
-----
train Loss: 0.0005 Acc: 0.9999
val Loss: 0.0068 Acc: 0.9981
Epoch 39/99
-----
train Loss: 0.0005 Acc: 0.9999
val Loss: 0.0084 Acc: 0.9969
Epoch 40/99
-----
train Loss: 0.0006 Acc: 0.9998
val Loss: 0.0074 Acc: 0.9980
Epoch 41/99
-----
train Loss: 0.0003 Acc: 1.0000
val Loss: 0.0081 Acc: 0.9971
```

Epoch 42/99

train Loss: 0.0005 Acc: 0.9998

val Loss: 0.0065 Acc: 0.9981

Epoch 43/99

train Loss: 0.0005 Acc: 0.9999

val Loss: 0.0056 Acc: 0.9985

Epoch 44/99

train Loss: 0.0004 Acc: 1.0000

val Loss: 0.0068 Acc: 0.9978

Epoch 45/99

train Loss: 0.0002 Acc: 1.0000

val Loss: 0.0074 Acc: 0.9981

Epoch 46/99

train Loss: 0.0004 Acc: 1.0000

val Loss: 0.0071 Acc: 0.9978

Epoch 47/99

train Loss: 0.0003 Acc: 1.0000

val Loss: 0.0077 Acc: 0.9978

Epoch 48/99

train Loss: 0.0003 Acc: 1.0000

val Loss: 0.0072 Acc: 0.9978

Epoch 49/99

train Loss: 0.0002 Acc: 1.0000

val Loss: 0.0089 Acc: 0.9968

Epoch 50/99

train Loss: 0.0004 Acc: 0.9999

val Loss: 0.0075 Acc: 0.9980

Epoch 51/99

train Loss: 0.0003 Acc: 1.0000

val Loss: 0.0073 Acc: 0.9981

Epoch 52/99

train Loss: 0.0003 Acc: 0.9999

val Loss: 0.0091 Acc: 0.9968

Epoch 53/99

train Loss: 0.0003 Acc: 0.9999

val Loss: 0.0074 Acc: 0.9976

Epoch 54/99

train Loss: 0.0004 Acc: 0.9999

val Loss: 0.0070 Acc: 0.9980

Epoch 55/99

train Loss: 0.0003 Acc: 0.9999

val Loss: 0.0077 Acc: 0.9978

Epoch 56/99

train Loss: 0.0003 Acc: 1.0000
val Loss: 0.0077 Acc: 0.9974
Epoch 57/99

train Loss: 0.0002 Acc: 1.0000
val Loss: 0.0069 Acc: 0.9983
Epoch 58/99

train Loss: 0.0002 Acc: 1.0000
val Loss: 0.0069 Acc: 0.9981
Epoch 59/99

train Loss: 0.0003 Acc: 0.9999
val Loss: 0.0083 Acc: 0.9971
Epoch 60/99

train Loss: 0.0002 Acc: 1.0000
val Loss: 0.0085 Acc: 0.9971
Epoch 61/99

train Loss: 0.0003 Acc: 1.0000
val Loss: 0.0074 Acc: 0.9976
Epoch 62/99

train Loss: 0.0003 Acc: 1.0000
val Loss: 0.0084 Acc: 0.9969
Epoch 63/99

train Loss: 0.0004 Acc: 0.9999
val Loss: 0.0072 Acc: 0.9980
Epoch 64/99

train Loss: 0.0004 Acc: 0.9999
val Loss: 0.0081 Acc: 0.9976
Epoch 65/99

train Loss: 0.0004 Acc: 0.9999
val Loss: 0.0106 Acc: 0.9966
Epoch 66/99

train Loss: 0.0003 Acc: 1.0000
val Loss: 0.0063 Acc: 0.9978
Epoch 67/99

train Loss: 0.0004 Acc: 0.9999
val Loss: 0.0070 Acc: 0.9980
Epoch 68/99

train Loss: 0.0005 Acc: 0.9998
val Loss: 0.0090 Acc: 0.9974
Epoch 69/99

train Loss: 0.0002 Acc: 1.0000
val Loss: 0.0063 Acc: 0.9980
Epoch 70/99

```
train Loss: 0.0004 Acc: 0.9999
val Loss: 0.0087 Acc: 0.9974
Epoch 71/99
-----
train Loss: 0.0003 Acc: 0.9999
val Loss: 0.0073 Acc: 0.9976
Epoch 72/99
-----
train Loss: 0.0005 Acc: 0.9999
val Loss: 0.0067 Acc: 0.9978
Epoch 73/99
-----
train Loss: 0.0002 Acc: 1.0000
val Loss: 0.0071 Acc: 0.9981
Epoch 74/99
-----
train Loss: 0.0003 Acc: 0.9999
val Loss: 0.0076 Acc: 0.9974
Epoch 75/99
-----
train Loss: 0.0002 Acc: 0.9999
val Loss: 0.0067 Acc: 0.9976
Epoch 76/99
-----
train Loss: 0.0002 Acc: 1.0000
val Loss: 0.0082 Acc: 0.9974
Epoch 77/99
-----
train Loss: 0.0003 Acc: 1.0000
val Loss: 0.0075 Acc: 0.9976
Epoch 78/99
-----
train Loss: 0.0002 Acc: 1.0000
val Loss: 0.0086 Acc: 0.9969
Epoch 79/99
-----
train Loss: 0.0002 Acc: 0.9999
val Loss: 0.0078 Acc: 0.9976
Epoch 80/99
-----
train Loss: 0.0002 Acc: 1.0000
val Loss: 0.0087 Acc: 0.9971
Epoch 81/99
-----
train Loss: 0.0002 Acc: 1.0000
val Loss: 0.0087 Acc: 0.9974
Epoch 82/99
-----
train Loss: 0.0005 Acc: 0.9999
val Loss: 0.0088 Acc: 0.9976
Epoch 83/99
-----
train Loss: 0.0002 Acc: 0.9999
val Loss: 0.0062 Acc: 0.9981
Epoch 84/99
-----
train Loss: 0.0002 Acc: 1.0000
```

```
val Loss: 0.0074 Acc: 0.9978
Epoch 85/99
-----
train Loss: 0.0002 Acc: 1.0000
val Loss: 0.0092 Acc: 0.9969
Epoch 86/99
-----
train Loss: 0.0003 Acc: 0.9999
val Loss: 0.0081 Acc: 0.9973
Epoch 87/99
-----
train Loss: 0.0004 Acc: 0.9999
val Loss: 0.0068 Acc: 0.9980
Epoch 88/99
-----
train Loss: 0.0002 Acc: 1.0000
val Loss: 0.0093 Acc: 0.9969
Epoch 89/99
-----
train Loss: 0.0002 Acc: 1.0000
val Loss: 0.0070 Acc: 0.9981
Epoch 90/99
-----
train Loss: 0.0004 Acc: 0.9999
val Loss: 0.0074 Acc: 0.9980
Epoch 91/99
-----
train Loss: 0.0004 Acc: 1.0000
val Loss: 0.0077 Acc: 0.9974
Epoch 92/99
-----
train Loss: 0.0007 Acc: 0.9998
val Loss: 0.0104 Acc: 0.9968
Epoch 93/99
-----
train Loss: 0.0003 Acc: 0.9999
val Loss: 0.0074 Acc: 0.9978
Epoch 94/99
-----
train Loss: 0.0004 Acc: 0.9998
val Loss: 0.0071 Acc: 0.9974
Epoch 95/99
-----
train Loss: 0.0002 Acc: 1.0000
val Loss: 0.0065 Acc: 0.9981
Epoch 96/99
-----
train Loss: 0.0005 Acc: 0.9998
val Loss: 0.0067 Acc: 0.9981
Epoch 97/99
-----
train Loss: 0.0005 Acc: 0.9999
val Loss: 0.0081 Acc: 0.9974
Epoch 98/99
-----
train Loss: 0.0003 Acc: 0.9999
val Loss: 0.0075 Acc: 0.9981
```

Epoch 99/99

train Loss: 0.0003 Acc: 0.9999
val Loss: 0.0073 Acc: 0.9976
Training complete
Best val Acc: 0.9985

```
In [ ]: # Print final accuracies for each learning rate
print(f"Final accuracy with learning rate 0.001: {epoch_acc:.4f}")
print(f"Final accuracy with learning rate 0.01: {best_acc_01:.4f}")
print(f"Final accuracy with learning rate 0.1: {best_acc_1:.4f}")

Final accuracy with learning rate 0.001: 0.9995
Final accuracy with learning rate 0.01: 1.0000
Final accuracy with learning rate 0.1: 0.9985
```

The best accuracy is achieved with the learning rate of **0.01**

1. Pretrained Model as Feature Extractor

a.

```
In [ ]: def train_feature_extractor(lr):
    # Load a pre-trained ResNet50 model
    model_extractor = models.resnet50(pretrained=True)

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

    # Replace the final fully connected layer with a new one with 2 output classes
    num_fts = model_extractor.fc.in_features
    model_extractor.fc = nn.Linear(num_fts, 2) # Assuming 2 classes for the Daimler Ped dataset
    model_extractor = model_extractor.to(device)

    # Define the criterion and optimizer (only for the last layer)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(model_extractor.fc.parameters(), lr=lr, momentum=0.9)

    # Use the same learning rate scheduler
    scheduler = MultiStepLR(optimizer, milestones=[25, 50, 75], gamma=0.1)

    # Train the model and return the best accuracy
    model_extractor, best_acc = train_model(model_extractor, criterion, optimizer, scheduler)
    return best_acc
```

```
In [ ]: best_acc_001 = train_feature_extractor(0.001)
        best_acc_01 = train_feature_extractor(0.01)
        best_acc_1 = train_feature_extractor(0.1)
```


Epoch 0/99

train Loss: 0.2801 Acc: 0.8927
val Loss: 0.1887 Acc: 0.9332
Epoch 1/99

train Loss: 0.1857 Acc: 0.9295
val Loss: 0.1640 Acc: 0.9401
Epoch 2/99

train Loss: 0.1684 Acc: 0.9361
val Loss: 0.1511 Acc: 0.9444
Epoch 3/99

train Loss: 0.1607 Acc: 0.9378
val Loss: 0.1473 Acc: 0.9468
Epoch 4/99

train Loss: 0.1564 Acc: 0.9401
val Loss: 0.1413 Acc: 0.9463
Epoch 5/99

train Loss: 0.1493 Acc: 0.9415
val Loss: 0.1370 Acc: 0.9509
Epoch 6/99

train Loss: 0.1431 Acc: 0.9446
val Loss: 0.1389 Acc: 0.9444
Epoch 7/99

train Loss: 0.1438 Acc: 0.9438
val Loss: 0.1315 Acc: 0.9544
Epoch 8/99

train Loss: 0.1412 Acc: 0.9442
val Loss: 0.1293 Acc: 0.9544
Epoch 9/99

train Loss: 0.1368 Acc: 0.9469
val Loss: 0.1246 Acc: 0.9551
Epoch 10/99

train Loss: 0.1344 Acc: 0.9496
val Loss: 0.1228 Acc: 0.9560
Epoch 11/99

train Loss: 0.1350 Acc: 0.9463
val Loss: 0.1217 Acc: 0.9524
Epoch 12/99

train Loss: 0.1310 Acc: 0.9498
val Loss: 0.1189 Acc: 0.9551
Epoch 13/99

train Loss: 0.1314 Acc: 0.9494
val Loss: 0.1184 Acc: 0.9561
Epoch 14/99

train Loss: 0.1271 Acc: 0.9505
val Loss: 0.1163 Acc: 0.9566
Epoch 15/99

train Loss: 0.1269 Acc: 0.9511
val Loss: 0.1157 Acc: 0.9575
Epoch 16/99

train Loss: 0.1275 Acc: 0.9503
val Loss: 0.1140 Acc: 0.9570
Epoch 17/99

train Loss: 0.1290 Acc: 0.9498
val Loss: 0.1125 Acc: 0.9573
Epoch 18/99

train Loss: 0.1283 Acc: 0.9509
val Loss: 0.1138 Acc: 0.9577
Epoch 19/99

train Loss: 0.1265 Acc: 0.9515
val Loss: 0.1135 Acc: 0.9568
Epoch 20/99

train Loss: 0.1253 Acc: 0.9522
val Loss: 0.1118 Acc: 0.9590
Epoch 21/99

train Loss: 0.1240 Acc: 0.9520
val Loss: 0.1195 Acc: 0.9575
Epoch 22/99

train Loss: 0.1217 Acc: 0.9534
val Loss: 0.1223 Acc: 0.9561
Epoch 23/99

train Loss: 0.1226 Acc: 0.9524
val Loss: 0.1176 Acc: 0.9532
Epoch 24/99

train Loss: 0.1176 Acc: 0.9554
val Loss: 0.1082 Acc: 0.9614
Epoch 25/99

train Loss: 0.1166 Acc: 0.9556
val Loss: 0.1077 Acc: 0.9609
Epoch 26/99

train Loss: 0.1157 Acc: 0.9563
val Loss: 0.1116 Acc: 0.9553
Epoch 27/99

train Loss: 0.1144 Acc: 0.9574
val Loss: 0.1080 Acc: 0.9600
Epoch 28/99

```
train Loss: 0.1156 Acc: 0.9552
val Loss: 0.1079 Acc: 0.9605
Epoch 29/99
-----
train Loss: 0.1150 Acc: 0.9572
val Loss: 0.1084 Acc: 0.9582
Epoch 30/99
-----
train Loss: 0.1117 Acc: 0.9573
val Loss: 0.1078 Acc: 0.9602
Epoch 31/99
-----
train Loss: 0.1171 Acc: 0.9557
val Loss: 0.1075 Acc: 0.9604
Epoch 32/99
-----
train Loss: 0.1180 Acc: 0.9542
val Loss: 0.1072 Acc: 0.9599
Epoch 33/99
-----
train Loss: 0.1143 Acc: 0.9572
val Loss: 0.1084 Acc: 0.9595
Epoch 34/99
-----
train Loss: 0.1113 Acc: 0.9585
val Loss: 0.1070 Acc: 0.9604
Epoch 35/99
-----
train Loss: 0.1157 Acc: 0.9572
val Loss: 0.1073 Acc: 0.9594
Epoch 36/99
-----
train Loss: 0.1167 Acc: 0.9554
val Loss: 0.1081 Acc: 0.9597
Epoch 37/99
-----
train Loss: 0.1152 Acc: 0.9556
val Loss: 0.1086 Acc: 0.9577
Epoch 38/99
-----
train Loss: 0.1143 Acc: 0.9568
val Loss: 0.1072 Acc: 0.9594
Epoch 39/99
-----
train Loss: 0.1184 Acc: 0.9547
val Loss: 0.1071 Acc: 0.9607
Epoch 40/99
-----
train Loss: 0.1134 Acc: 0.9562
val Loss: 0.1069 Acc: 0.9595
Epoch 41/99
-----
train Loss: 0.1166 Acc: 0.9543
val Loss: 0.1069 Acc: 0.9590
Epoch 42/99
-----
train Loss: 0.1145 Acc: 0.9555
```

```
val Loss: 0.1065 Acc: 0.9602
Epoch 43/99
-----
train Loss: 0.1143 Acc: 0.9578
val Loss: 0.1073 Acc: 0.9587
Epoch 44/99
-----
train Loss: 0.1143 Acc: 0.9551
val Loss: 0.1070 Acc: 0.9616
Epoch 45/99
-----
train Loss: 0.1140 Acc: 0.9562
val Loss: 0.1063 Acc: 0.9609
Epoch 46/99
-----
train Loss: 0.1163 Acc: 0.9573
val Loss: 0.1081 Acc: 0.9585
Epoch 47/99
-----
train Loss: 0.1185 Acc: 0.9550
val Loss: 0.1065 Acc: 0.9600
Epoch 48/99
-----
train Loss: 0.1166 Acc: 0.9534
val Loss: 0.1060 Acc: 0.9607
Epoch 49/99
-----
train Loss: 0.1116 Acc: 0.9600
val Loss: 0.1056 Acc: 0.9617
Epoch 50/99
-----
train Loss: 0.1134 Acc: 0.9566
val Loss: 0.1070 Acc: 0.9605
Epoch 51/99
-----
train Loss: 0.1133 Acc: 0.9568
val Loss: 0.1063 Acc: 0.9605
Epoch 52/99
-----
train Loss: 0.1129 Acc: 0.9571
val Loss: 0.1071 Acc: 0.9609
Epoch 53/99
-----
train Loss: 0.1159 Acc: 0.9560
val Loss: 0.1073 Acc: 0.9587
Epoch 54/99
-----
train Loss: 0.1131 Acc: 0.9582
val Loss: 0.1064 Acc: 0.9602
Epoch 55/99
-----
train Loss: 0.1131 Acc: 0.9563
val Loss: 0.1088 Acc: 0.9602
Epoch 56/99
-----
train Loss: 0.1141 Acc: 0.9577
val Loss: 0.1067 Acc: 0.9612
```

Epoch 57/99

train Loss: 0.1145 Acc: 0.9572

val Loss: 0.1066 Acc: 0.9612

Epoch 58/99

train Loss: 0.1135 Acc: 0.9569

val Loss: 0.1067 Acc: 0.9609

Epoch 59/99

train Loss: 0.1134 Acc: 0.9565

val Loss: 0.1065 Acc: 0.9597

Epoch 60/99

train Loss: 0.1137 Acc: 0.9568

val Loss: 0.1064 Acc: 0.9605

Epoch 61/99

train Loss: 0.1149 Acc: 0.9565

val Loss: 0.1082 Acc: 0.9583

Epoch 62/99

train Loss: 0.1140 Acc: 0.9570

val Loss: 0.1061 Acc: 0.9612

Epoch 63/99

train Loss: 0.1126 Acc: 0.9573

val Loss: 0.1063 Acc: 0.9607

Epoch 64/99

train Loss: 0.1131 Acc: 0.9574

val Loss: 0.1061 Acc: 0.9602

Epoch 65/99

train Loss: 0.1164 Acc: 0.9557

val Loss: 0.1064 Acc: 0.9614

Epoch 66/99

train Loss: 0.1114 Acc: 0.9578

val Loss: 0.1053 Acc: 0.9617

Epoch 67/99

train Loss: 0.1144 Acc: 0.9567

val Loss: 0.1065 Acc: 0.9600

Epoch 68/99

train Loss: 0.1118 Acc: 0.9582

val Loss: 0.1066 Acc: 0.9600

Epoch 69/99

train Loss: 0.1143 Acc: 0.9557

val Loss: 0.1093 Acc: 0.9573

Epoch 70/99

train Loss: 0.1132 Acc: 0.9574

val Loss: 0.1074 Acc: 0.9600

Epoch 71/99

train Loss: 0.1139 Acc: 0.9568
val Loss: 0.1081 Acc: 0.9590
Epoch 72/99

train Loss: 0.1126 Acc: 0.9579
val Loss: 0.1060 Acc: 0.9607
Epoch 73/99

train Loss: 0.1164 Acc: 0.9567
val Loss: 0.1064 Acc: 0.9611
Epoch 74/99

train Loss: 0.1139 Acc: 0.9557
val Loss: 0.1058 Acc: 0.9607
Epoch 75/99

train Loss: 0.1122 Acc: 0.9575
val Loss: 0.1059 Acc: 0.9617
Epoch 76/99

train Loss: 0.1160 Acc: 0.9557
val Loss: 0.1064 Acc: 0.9604
Epoch 77/99

train Loss: 0.1137 Acc: 0.9566
val Loss: 0.1069 Acc: 0.9597
Epoch 78/99

train Loss: 0.1161 Acc: 0.9556
val Loss: 0.1067 Acc: 0.9599
Epoch 79/99

train Loss: 0.1125 Acc: 0.9572
val Loss: 0.1065 Acc: 0.9609
Epoch 80/99

train Loss: 0.1162 Acc: 0.9562
val Loss: 0.1058 Acc: 0.9617
Epoch 81/99

train Loss: 0.1124 Acc: 0.9579
val Loss: 0.1069 Acc: 0.9594
Epoch 82/99

train Loss: 0.1125 Acc: 0.9555
val Loss: 0.1061 Acc: 0.9617
Epoch 83/99

train Loss: 0.1125 Acc: 0.9553
val Loss: 0.1057 Acc: 0.9607
Epoch 84/99

train Loss: 0.1107 Acc: 0.9579
val Loss: 0.1059 Acc: 0.9617
Epoch 85/99

```
train Loss: 0.1101 Acc: 0.9593
val Loss: 0.1057 Acc: 0.9616
Epoch 86/99
-----
train Loss: 0.1150 Acc: 0.9569
val Loss: 0.1063 Acc: 0.9609
Epoch 87/99
-----
train Loss: 0.1133 Acc: 0.9571
val Loss: 0.1080 Acc: 0.9587
Epoch 88/99
-----
train Loss: 0.1135 Acc: 0.9565
val Loss: 0.1065 Acc: 0.9612
Epoch 89/99
-----
train Loss: 0.1129 Acc: 0.9569
val Loss: 0.1068 Acc: 0.9587
Epoch 90/99
-----
train Loss: 0.1140 Acc: 0.9569
val Loss: 0.1063 Acc: 0.9605
Epoch 91/99
-----
train Loss: 0.1134 Acc: 0.9567
val Loss: 0.1094 Acc: 0.9571
Epoch 92/99
-----
train Loss: 0.1136 Acc: 0.9572
val Loss: 0.1059 Acc: 0.9621
Epoch 93/99
-----
train Loss: 0.1125 Acc: 0.9572
val Loss: 0.1059 Acc: 0.9614
Epoch 94/99
-----
train Loss: 0.1139 Acc: 0.9558
val Loss: 0.1061 Acc: 0.9607
Epoch 95/99
-----
train Loss: 0.1120 Acc: 0.9589
val Loss: 0.1053 Acc: 0.9611
Epoch 96/99
-----
train Loss: 0.1128 Acc: 0.9585
val Loss: 0.1058 Acc: 0.9602
Epoch 97/99
-----
train Loss: 0.1105 Acc: 0.9565
val Loss: 0.1074 Acc: 0.9602
Epoch 98/99
-----
train Loss: 0.1137 Acc: 0.9561
val Loss: 0.1058 Acc: 0.9595
Epoch 99/99
-----
train Loss: 0.1103 Acc: 0.9575
```

```
val Loss: 0.1066 Acc: 0.9602
Training complete
Best val Acc: 0.9621
Epoch 0/99
-----
train Loss: 0.2233 Acc: 0.9103
val Loss: 0.1651 Acc: 0.9386
Epoch 1/99
-----
train Loss: 0.1905 Acc: 0.9293
val Loss: 0.1132 Acc: 0.9577
Epoch 2/99
-----
train Loss: 0.1494 Acc: 0.9403
val Loss: 0.1056 Acc: 0.9614
Epoch 3/99
-----
train Loss: 0.1364 Acc: 0.9460
val Loss: 0.1046 Acc: 0.9614
Epoch 4/99
-----
train Loss: 0.1440 Acc: 0.9443
val Loss: 0.1008 Acc: 0.9614
Epoch 5/99
-----
train Loss: 0.1413 Acc: 0.9461
val Loss: 0.1014 Acc: 0.9607
Epoch 6/99
-----
train Loss: 0.1709 Acc: 0.9373
val Loss: 0.0958 Acc: 0.9636
Epoch 7/99
-----
train Loss: 0.1309 Acc: 0.9501
val Loss: 0.1069 Acc: 0.9631
Epoch 8/99
-----
train Loss: 0.1329 Acc: 0.9486
val Loss: 0.1312 Acc: 0.9457
Epoch 9/99
-----
train Loss: 0.1166 Acc: 0.9557
val Loss: 0.0910 Acc: 0.9626
Epoch 10/99
-----
train Loss: 0.1308 Acc: 0.9493
val Loss: 0.1018 Acc: 0.9626
Epoch 11/99
-----
train Loss: 0.1184 Acc: 0.9555
val Loss: 0.0955 Acc: 0.9653
Epoch 12/99
-----
train Loss: 0.1129 Acc: 0.9563
val Loss: 0.1020 Acc: 0.9578
Epoch 13/99
-----
```



```
train Loss: 0.1071 Acc: 0.9588
val Loss: 0.1160 Acc: 0.9512
Epoch 14/99
-----
train Loss: 0.1147 Acc: 0.9575
val Loss: 0.0824 Acc: 0.9687
Epoch 15/99
-----
train Loss: 0.1186 Acc: 0.9554
val Loss: 0.1058 Acc: 0.9554
Epoch 16/99
-----
train Loss: 0.1201 Acc: 0.9544
val Loss: 0.1042 Acc: 0.9650
Epoch 17/99
-----
train Loss: 0.1120 Acc: 0.9569
val Loss: 0.1550 Acc: 0.9437
Epoch 18/99
-----
train Loss: 0.0986 Acc: 0.9622
val Loss: 0.0881 Acc: 0.9682
Epoch 19/99
-----
train Loss: 0.1047 Acc: 0.9595
val Loss: 0.0835 Acc: 0.9670
Epoch 20/99
-----
train Loss: 0.1003 Acc: 0.9605
val Loss: 0.0999 Acc: 0.9590
Epoch 21/99
-----
train Loss: 0.1006 Acc: 0.9613
val Loss: 0.0791 Acc: 0.9675
Epoch 22/99
-----
train Loss: 0.1060 Acc: 0.9578
val Loss: 0.0777 Acc: 0.9689
Epoch 23/99
-----
train Loss: 0.1059 Acc: 0.9593
val Loss: 0.0956 Acc: 0.9655
Epoch 24/99
-----
train Loss: 0.1137 Acc: 0.9585
val Loss: 0.0771 Acc: 0.9694
Epoch 25/99
-----
train Loss: 0.0769 Acc: 0.9702
val Loss: 0.0790 Acc: 0.9665
Epoch 26/99
-----
train Loss: 0.0776 Acc: 0.9707
val Loss: 0.0741 Acc: 0.9709
Epoch 27/99
-----
train Loss: 0.0802 Acc: 0.9700
```

```
val Loss: 0.0731 Acc: 0.9701
Epoch 28/99
-----
train Loss: 0.0750 Acc: 0.9724
val Loss: 0.0719 Acc: 0.9699
Epoch 29/99
-----
train Loss: 0.0725 Acc: 0.9719
val Loss: 0.0721 Acc: 0.9704
Epoch 30/99
-----
train Loss: 0.0746 Acc: 0.9716
val Loss: 0.0756 Acc: 0.9689
Epoch 31/99
-----
train Loss: 0.0745 Acc: 0.9724
val Loss: 0.0719 Acc: 0.9697
Epoch 32/99
-----
train Loss: 0.0764 Acc: 0.9711
val Loss: 0.0711 Acc: 0.9704
Epoch 33/99
-----
train Loss: 0.0752 Acc: 0.9707
val Loss: 0.0715 Acc: 0.9713
Epoch 34/99
-----
train Loss: 0.0777 Acc: 0.9702
val Loss: 0.0769 Acc: 0.9682
Epoch 35/99
-----
train Loss: 0.0734 Acc: 0.9722
val Loss: 0.0702 Acc: 0.9718
Epoch 36/99
-----
train Loss: 0.0757 Acc: 0.9711
val Loss: 0.0733 Acc: 0.9699
Epoch 37/99
-----
train Loss: 0.0763 Acc: 0.9702
val Loss: 0.0710 Acc: 0.9704
Epoch 38/99
-----
train Loss: 0.0755 Acc: 0.9705
val Loss: 0.0709 Acc: 0.9707
Epoch 39/99
-----
train Loss: 0.0744 Acc: 0.9713
val Loss: 0.0711 Acc: 0.9704
Epoch 40/99
-----
train Loss: 0.0746 Acc: 0.9705
val Loss: 0.0705 Acc: 0.9723
Epoch 41/99
-----
train Loss: 0.0790 Acc: 0.9690
val Loss: 0.0704 Acc: 0.9706
```

Epoch 42/99

train Loss: 0.0767 Acc: 0.9700

val Loss: 0.0698 Acc: 0.9723

Epoch 43/99

train Loss: 0.0782 Acc: 0.9691

val Loss: 0.0703 Acc: 0.9704

Epoch 44/99

train Loss: 0.0746 Acc: 0.9722

val Loss: 0.0704 Acc: 0.9711

Epoch 45/99

train Loss: 0.0738 Acc: 0.9716

val Loss: 0.0697 Acc: 0.9716

Epoch 46/99

train Loss: 0.0749 Acc: 0.9715

val Loss: 0.0705 Acc: 0.9711

Epoch 47/99

train Loss: 0.0707 Acc: 0.9728

val Loss: 0.0720 Acc: 0.9692

Epoch 48/99

train Loss: 0.0765 Acc: 0.9705

val Loss: 0.0714 Acc: 0.9697

Epoch 49/99

train Loss: 0.0766 Acc: 0.9707

val Loss: 0.0772 Acc: 0.9704

Epoch 50/99

train Loss: 0.0724 Acc: 0.9727

val Loss: 0.0708 Acc: 0.9704

Epoch 51/99

train Loss: 0.0718 Acc: 0.9728

val Loss: 0.0705 Acc: 0.9707

Epoch 52/99

train Loss: 0.0692 Acc: 0.9730

val Loss: 0.0698 Acc: 0.9719

Epoch 53/99

train Loss: 0.0720 Acc: 0.9726

val Loss: 0.0709 Acc: 0.9699

Epoch 54/99

train Loss: 0.0744 Acc: 0.9723

val Loss: 0.0700 Acc: 0.9721

Epoch 55/99

train Loss: 0.0715 Acc: 0.9725

val Loss: 0.0710 Acc: 0.9713

Epoch 56/99

train Loss: 0.0676 Acc: 0.9740
val Loss: 0.0694 Acc: 0.9714
Epoch 57/99

train Loss: 0.0730 Acc: 0.9723
val Loss: 0.0700 Acc: 0.9719
Epoch 58/99

train Loss: 0.0699 Acc: 0.9733
val Loss: 0.0715 Acc: 0.9731
Epoch 59/99

train Loss: 0.0724 Acc: 0.9722
val Loss: 0.0706 Acc: 0.9730
Epoch 60/99

train Loss: 0.0724 Acc: 0.9722
val Loss: 0.0702 Acc: 0.9713
Epoch 61/99

train Loss: 0.0723 Acc: 0.9731
val Loss: 0.0707 Acc: 0.9714
Epoch 62/99

train Loss: 0.0728 Acc: 0.9733
val Loss: 0.0713 Acc: 0.9706
Epoch 63/99

train Loss: 0.0753 Acc: 0.9710
val Loss: 0.0744 Acc: 0.9716
Epoch 64/99

train Loss: 0.0711 Acc: 0.9732
val Loss: 0.0702 Acc: 0.9716
Epoch 65/99

train Loss: 0.0699 Acc: 0.9750
val Loss: 0.0699 Acc: 0.9733
Epoch 66/99

train Loss: 0.0723 Acc: 0.9727
val Loss: 0.0699 Acc: 0.9718
Epoch 67/99

train Loss: 0.0727 Acc: 0.9725
val Loss: 0.0691 Acc: 0.9718
Epoch 68/99

train Loss: 0.0737 Acc: 0.9710
val Loss: 0.0695 Acc: 0.9714
Epoch 69/99

train Loss: 0.0708 Acc: 0.9735
val Loss: 0.0698 Acc: 0.9709
Epoch 70/99

```
train Loss: 0.0740 Acc: 0.9714
val Loss: 0.0705 Acc: 0.9707
Epoch 71/99
-----
train Loss: 0.0716 Acc: 0.9720
val Loss: 0.0693 Acc: 0.9726
Epoch 72/99
-----
train Loss: 0.0722 Acc: 0.9735
val Loss: 0.0707 Acc: 0.9713
Epoch 73/99
-----
train Loss: 0.0719 Acc: 0.9734
val Loss: 0.0704 Acc: 0.9716
Epoch 74/99
-----
train Loss: 0.0743 Acc: 0.9721
val Loss: 0.0699 Acc: 0.9718
Epoch 75/99
-----
train Loss: 0.0714 Acc: 0.9736
val Loss: 0.0708 Acc: 0.9713
Epoch 76/99
-----
train Loss: 0.0726 Acc: 0.9733
val Loss: 0.0700 Acc: 0.9709
Epoch 77/99
-----
train Loss: 0.0724 Acc: 0.9718
val Loss: 0.0692 Acc: 0.9714
Epoch 78/99
-----
train Loss: 0.0733 Acc: 0.9726
val Loss: 0.0709 Acc: 0.9706
Epoch 79/99
-----
train Loss: 0.0740 Acc: 0.9721
val Loss: 0.0715 Acc: 0.9711
Epoch 80/99
-----
train Loss: 0.0708 Acc: 0.9731
val Loss: 0.0731 Acc: 0.9731
Epoch 81/99
-----
train Loss: 0.0713 Acc: 0.9726
val Loss: 0.0712 Acc: 0.9711
Epoch 82/99
-----
train Loss: 0.0715 Acc: 0.9722
val Loss: 0.0700 Acc: 0.9719
Epoch 83/99
-----
train Loss: 0.0705 Acc: 0.9737
val Loss: 0.0702 Acc: 0.9704
Epoch 84/99
-----
train Loss: 0.0716 Acc: 0.9724
```

```
val Loss: 0.0702 Acc: 0.9713
Epoch 85/99
-----
train Loss: 0.0744 Acc: 0.9729
val Loss: 0.0707 Acc: 0.9718
Epoch 86/99
-----
train Loss: 0.0727 Acc: 0.9721
val Loss: 0.0703 Acc: 0.9716
Epoch 87/99
-----
train Loss: 0.0705 Acc: 0.9735
val Loss: 0.0692 Acc: 0.9718
Epoch 88/99
-----
train Loss: 0.0722 Acc: 0.9725
val Loss: 0.0708 Acc: 0.9719
Epoch 89/99
-----
train Loss: 0.0721 Acc: 0.9727
val Loss: 0.0697 Acc: 0.9716
Epoch 90/99
-----
train Loss: 0.0687 Acc: 0.9753
val Loss: 0.0697 Acc: 0.9718
Epoch 91/99
-----
train Loss: 0.0720 Acc: 0.9740
val Loss: 0.0699 Acc: 0.9713
Epoch 92/99
-----
train Loss: 0.0694 Acc: 0.9734
val Loss: 0.0750 Acc: 0.9696
Epoch 93/99
-----
train Loss: 0.0708 Acc: 0.9730
val Loss: 0.0701 Acc: 0.9714
Epoch 94/99
-----
train Loss: 0.0705 Acc: 0.9730
val Loss: 0.0695 Acc: 0.9719
Epoch 95/99
-----
train Loss: 0.0729 Acc: 0.9729
val Loss: 0.0700 Acc: 0.9699
Epoch 96/99
-----
train Loss: 0.0723 Acc: 0.9724
val Loss: 0.0704 Acc: 0.9723
Epoch 97/99
-----
train Loss: 0.0714 Acc: 0.9730
val Loss: 0.0710 Acc: 0.9701
Epoch 98/99
-----
train Loss: 0.0736 Acc: 0.9726
val Loss: 0.0695 Acc: 0.9709
```

Epoch 99/99

train Loss: 0.0727 Acc: 0.9714
val Loss: 0.0701 Acc: 0.9709
Training complete
Best val Acc: 0.9733
Epoch 0/99

train Loss: 1.9501 Acc: 0.8873
val Loss: 0.6180 Acc: 0.9476
Epoch 1/99

train Loss: 1.0960 Acc: 0.9200
val Loss: 0.6377 Acc: 0.9507
Epoch 2/99

train Loss: 0.8402 Acc: 0.9303
val Loss: 4.8693 Acc: 0.7728
Epoch 3/99

train Loss: 0.9999 Acc: 0.9293
val Loss: 0.4672 Acc: 0.9587
Epoch 4/99

train Loss: 0.8441 Acc: 0.9327
val Loss: 1.8212 Acc: 0.8845
Epoch 5/99

train Loss: 0.8115 Acc: 0.9347
val Loss: 0.3599 Acc: 0.9639
Epoch 6/99

train Loss: 0.6408 Acc: 0.9443
val Loss: 1.2538 Acc: 0.8917
Epoch 7/99

train Loss: 0.7515 Acc: 0.9378
val Loss: 1.2011 Acc: 0.9019
Epoch 8/99

train Loss: 0.8678 Acc: 0.9348
val Loss: 1.1148 Acc: 0.9090
Epoch 9/99

train Loss: 0.7112 Acc: 0.9407
val Loss: 1.1090 Acc: 0.9182
Epoch 10/99

train Loss: 0.6080 Acc: 0.9466
val Loss: 0.5483 Acc: 0.9563
Epoch 11/99

train Loss: 0.8219 Acc: 0.9379
val Loss: 4.5555 Acc: 0.8005
Epoch 12/99

train Loss: 0.9631 Acc: 0.9372

```
val Loss: 0.4711 Acc: 0.9638
Epoch 13/99
-----
train Loss: 0.6191 Acc: 0.9500
val Loss: 0.4629 Acc: 0.9633
Epoch 14/99
-----
train Loss: 0.8113 Acc: 0.9403
val Loss: 0.3898 Acc: 0.9612
Epoch 15/99
-----
train Loss: 0.5887 Acc: 0.9497
val Loss: 1.2928 Acc: 0.9046
Epoch 16/99
-----
train Loss: 0.5995 Acc: 0.9500
val Loss: 0.4585 Acc: 0.9622
Epoch 17/99
-----
train Loss: 0.8778 Acc: 0.9403
val Loss: 0.8047 Acc: 0.9446
Epoch 18/99
-----
train Loss: 0.6947 Acc: 0.9480
val Loss: 3.1523 Acc: 0.8429
Epoch 19/99
-----
train Loss: 0.5937 Acc: 0.9519
val Loss: 0.6561 Acc: 0.9495
Epoch 20/99
-----
train Loss: 0.5270 Acc: 0.9545
val Loss: 0.2825 Acc: 0.9668
Epoch 21/99
-----
train Loss: 1.0519 Acc: 0.9355
val Loss: 0.4783 Acc: 0.9650
Epoch 22/99
-----
train Loss: 0.5721 Acc: 0.9572
val Loss: 0.3292 Acc: 0.9650
Epoch 23/99
-----
train Loss: 0.6250 Acc: 0.9504
val Loss: 0.5810 Acc: 0.9531
Epoch 24/99
-----
train Loss: 0.7681 Acc: 0.9463
val Loss: 0.6347 Acc: 0.9546
Epoch 25/99
-----
train Loss: 0.2984 Acc: 0.9703
val Loss: 0.3438 Acc: 0.9651
Epoch 26/99
-----
train Loss: 0.3083 Acc: 0.9701
val Loss: 0.2881 Acc: 0.9690
```


Epoch 27/99

train Loss: 0.2981 Acc: 0.9707
val Loss: 0.2689 Acc: 0.9699
Epoch 28/99

train Loss: 0.2769 Acc: 0.9700
val Loss: 0.2552 Acc: 0.9706
Epoch 29/99

train Loss: 0.2465 Acc: 0.9725
val Loss: 0.2431 Acc: 0.9716
Epoch 30/99

train Loss: 0.2424 Acc: 0.9731
val Loss: 0.2350 Acc: 0.9713
Epoch 31/99

train Loss: 0.2379 Acc: 0.9724
val Loss: 0.2492 Acc: 0.9684
Epoch 32/99

train Loss: 0.2603 Acc: 0.9702
val Loss: 0.2233 Acc: 0.9711
Epoch 33/99

train Loss: 0.2206 Acc: 0.9730
val Loss: 0.2308 Acc: 0.9704
Epoch 34/99

train Loss: 0.2347 Acc: 0.9709
val Loss: 0.2694 Acc: 0.9702
Epoch 35/99

train Loss: 0.2362 Acc: 0.9708
val Loss: 0.2272 Acc: 0.9709
Epoch 36/99

train Loss: 0.2194 Acc: 0.9729
val Loss: 0.2252 Acc: 0.9713
Epoch 37/99

train Loss: 0.2037 Acc: 0.9738
val Loss: 0.1964 Acc: 0.9721
Epoch 38/99

train Loss: 0.2254 Acc: 0.9705
val Loss: 0.1922 Acc: 0.9721
Epoch 39/99

train Loss: 0.2081 Acc: 0.9716
val Loss: 0.1922 Acc: 0.9713
Epoch 40/99

train Loss: 0.2255 Acc: 0.9698
val Loss: 0.1854 Acc: 0.9718
Epoch 41/99

train Loss: 0.2076 Acc: 0.9738
val Loss: 0.2067 Acc: 0.9704
Epoch 42/99

train Loss: 0.2208 Acc: 0.9703
val Loss: 0.2344 Acc: 0.9713
Epoch 43/99

train Loss: 0.1854 Acc: 0.9738
val Loss: 0.1809 Acc: 0.9721
Epoch 44/99

train Loss: 0.1867 Acc: 0.9732
val Loss: 0.1856 Acc: 0.9706
Epoch 45/99

train Loss: 0.2130 Acc: 0.9702
val Loss: 0.1790 Acc: 0.9730
Epoch 46/99

train Loss: 0.1940 Acc: 0.9725
val Loss: 0.2229 Acc: 0.9699
Epoch 47/99

train Loss: 0.1833 Acc: 0.9727
val Loss: 0.1751 Acc: 0.9724
Epoch 48/99

train Loss: 0.1819 Acc: 0.9729
val Loss: 0.1747 Acc: 0.9713
Epoch 49/99

train Loss: 0.2011 Acc: 0.9719
val Loss: 0.1854 Acc: 0.9714
Epoch 50/99

train Loss: 0.1677 Acc: 0.9745
val Loss: 0.1705 Acc: 0.9731
Epoch 51/99

train Loss: 0.1650 Acc: 0.9756
val Loss: 0.1672 Acc: 0.9724
Epoch 52/99

train Loss: 0.1741 Acc: 0.9738
val Loss: 0.1670 Acc: 0.9726
Epoch 53/99

train Loss: 0.1653 Acc: 0.9744
val Loss: 0.1647 Acc: 0.9735
Epoch 54/99

train Loss: 0.1550 Acc: 0.9764
val Loss: 0.1601 Acc: 0.9736
Epoch 55/99

```
train Loss: 0.1660 Acc: 0.9751
val Loss: 0.1650 Acc: 0.9723
Epoch 56/99
-----
train Loss: 0.1713 Acc: 0.9731
val Loss: 0.1666 Acc: 0.9723
Epoch 57/99
-----
train Loss: 0.1573 Acc: 0.9768
val Loss: 0.1675 Acc: 0.9719
Epoch 58/99
-----
train Loss: 0.1607 Acc: 0.9744
val Loss: 0.1681 Acc: 0.9733
Epoch 59/99
-----
train Loss: 0.1737 Acc: 0.9738
val Loss: 0.1644 Acc: 0.9724
Epoch 60/99
-----
train Loss: 0.1658 Acc: 0.9749
val Loss: 0.1629 Acc: 0.9723
Epoch 61/99
-----
train Loss: 0.1783 Acc: 0.9733
val Loss: 0.1762 Acc: 0.9721
Epoch 62/99
-----
train Loss: 0.1623 Acc: 0.9754
val Loss: 0.1703 Acc: 0.9719
Epoch 63/99
-----
train Loss: 0.1628 Acc: 0.9743
val Loss: 0.1602 Acc: 0.9730
Epoch 64/99
-----
train Loss: 0.1572 Acc: 0.9758
val Loss: 0.1651 Acc: 0.9731
Epoch 65/99
-----
train Loss: 0.1643 Acc: 0.9742
val Loss: 0.1591 Acc: 0.9738
Epoch 66/99
-----
train Loss: 0.1573 Acc: 0.9748
val Loss: 0.1644 Acc: 0.9723
Epoch 67/99
-----
train Loss: 0.1688 Acc: 0.9738
val Loss: 0.1789 Acc: 0.9726
Epoch 68/99
-----
train Loss: 0.1540 Acc: 0.9744
val Loss: 0.1630 Acc: 0.9726
Epoch 69/99
-----
train Loss: 0.1707 Acc: 0.9736
```

```
val Loss: 0.1601 Acc: 0.9733
Epoch 70/99
-----
train Loss: 0.1662 Acc: 0.9748
val Loss: 0.1640 Acc: 0.9713
Epoch 71/99
-----
train Loss: 0.1567 Acc: 0.9742
val Loss: 0.1591 Acc: 0.9736
Epoch 72/99
-----
train Loss: 0.1534 Acc: 0.9757
val Loss: 0.1696 Acc: 0.9735
Epoch 73/99
-----
train Loss: 0.1681 Acc: 0.9737
val Loss: 0.1704 Acc: 0.9730
Epoch 74/99
-----
train Loss: 0.1623 Acc: 0.9744
val Loss: 0.1619 Acc: 0.9743
Epoch 75/99
-----
train Loss: 0.1602 Acc: 0.9750
val Loss: 0.1593 Acc: 0.9738
Epoch 76/99
-----
train Loss: 0.1660 Acc: 0.9739
val Loss: 0.1636 Acc: 0.9721
Epoch 77/99
-----
train Loss: 0.1831 Acc: 0.9737
val Loss: 0.1639 Acc: 0.9728
Epoch 78/99
-----
train Loss: 0.1656 Acc: 0.9744
val Loss: 0.1729 Acc: 0.9731
Epoch 79/99
-----
train Loss: 0.1609 Acc: 0.9754
val Loss: 0.1706 Acc: 0.9719
Epoch 80/99
-----
train Loss: 0.1599 Acc: 0.9754
val Loss: 0.1612 Acc: 0.9730
Epoch 81/99
-----
train Loss: 0.1586 Acc: 0.9750
val Loss: 0.1602 Acc: 0.9730
Epoch 82/99
-----
train Loss: 0.1570 Acc: 0.9747
val Loss: 0.1709 Acc: 0.9726
Epoch 83/99
-----
train Loss: 0.1635 Acc: 0.9741
val Loss: 0.1626 Acc: 0.9741
```

Epoch 84/99

train Loss: 0.1681 Acc: 0.9747
val Loss: 0.1683 Acc: 0.9723
Epoch 85/99

train Loss: 0.1539 Acc: 0.9759
val Loss: 0.1548 Acc: 0.9738
Epoch 86/99

train Loss: 0.1578 Acc: 0.9753
val Loss: 0.1594 Acc: 0.9743
Epoch 87/99

train Loss: 0.1662 Acc: 0.9758
val Loss: 0.1635 Acc: 0.9736
Epoch 88/99

train Loss: 0.1717 Acc: 0.9741
val Loss: 0.1637 Acc: 0.9735
Epoch 89/99

train Loss: 0.1440 Acc: 0.9752
val Loss: 0.1659 Acc: 0.9728
Epoch 90/99

train Loss: 0.1609 Acc: 0.9758
val Loss: 0.1655 Acc: 0.9733
Epoch 91/99

train Loss: 0.1570 Acc: 0.9753
val Loss: 0.1785 Acc: 0.9726
Epoch 92/99

train Loss: 0.1603 Acc: 0.9751
val Loss: 0.1550 Acc: 0.9735
Epoch 93/99

train Loss: 0.1517 Acc: 0.9751
val Loss: 0.1657 Acc: 0.9730
Epoch 94/99

train Loss: 0.1503 Acc: 0.9760
val Loss: 0.1631 Acc: 0.9736
Epoch 95/99

train Loss: 0.1575 Acc: 0.9747
val Loss: 0.1573 Acc: 0.9731
Epoch 96/99

train Loss: 0.1459 Acc: 0.9765
val Loss: 0.1653 Acc: 0.9721
Epoch 97/99

train Loss: 0.1614 Acc: 0.9756
val Loss: 0.1639 Acc: 0.9735
Epoch 98/99

```
-----  
train Loss: 0.1613 Acc: 0.9752  
val Loss: 0.1739 Acc: 0.9714  
Epoch 99/99  
-----  
train Loss: 0.1582 Acc: 0.9746  
val Loss: 0.1636 Acc: 0.9735  
Training complete  
Best val Acc: 0.9743
```

```
In [ ]: print(f"Final accuracy with learning rate 0.001: {best_acc_001:.4f}")  
        print(f"Final accuracy with learning rate 0.01: {best_acc_01:.4f}")  
        print(f"Final accuracy with learning rate 0.1: {best_acc_1:.4f}")  
  
Final accuracy with learning rate 0.001: 0.9621  
Final accuracy with learning rate 0.01: 0.9733  
Final accuracy with learning rate 0.1: 0.9743
```

The best performance was achieved with the **finetuning** approach and a learning rate of **0.01**, yielding a perfect accuracy of 1.0000 on the target dataset. For feature extraction, the best performance was with a learning rate of 0.1, achieving 0.9743 accuracy.

I believe that the fine-tuning approach yielded better performance than the feature extraction method because all layers of the pretrained model were adapted to better predict the output classes. This network-wide finetuning resulted in a network where each layer in the network can more accurately capture and refine features specific to the Daimler Ped dataset.

I believe that the learning rate of 0.01 was large enough to allow for fast convergence in 100 epochs, while not overshooting and causing major oscillations in the loss.

While the models trained through feature extraction performed worse than the finetuned models, training was greatly expedited to the point where it becomes a viable option if training time restraints compete with accuracy (maybe in situations where new models have to be trained frequently to prevent model drift).

Problem 3

1. Dataset Setup

```
In [1]: from datasets import load_dataset
import random

# Load the GSM8K dataset from Hugging Face
gsm8k = load_dataset("gsm8k", "main", split="test")

# Randomly select one question from the test set
random_question = gsm8k[random.randint(0, len(gsm8k) - 1)]

# Display the selected question
print("Selected Question:")
print("Question:", random_question['question'])
print("Answer:", random_question['answer'])
```

/Users/nathancoulibaly/Documents/Columbia_Grad/LLM & DL System Performance/HW3/pytorch-ssd/.venv/lib/python3.12/site-packages/tqdm/auto.py:21: TqdmWarning: IProgress not found. Please update jupyter and ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/user_install.html
 from .autonotebook import tqdm as notebook_tqdm

Selected Question:

Question: Alex, Stan, and Adewolfe are trying to catch them all, Pokemon that is. Together they have caught 339 Pokemon. Alex has caught 5 more than Stan, and Stan has caught 13 less than 4 times as many as Adewolfe has caught. How many Pokemon has Stan caught?

Answer: Let x represent the number of Pokemon Adewolfe caught

Stan: $4x-13$

Alex: $5+(4x-13)=4x-8$

Total: $x+4x-13+4x-8=339$

$9x-21=339$

$9x=360$

$x=40=40$ Pokemon

Stan: $4(40)-13=147$ Pokemon

147

1. Model Selection: I will be using the Gemini API with gemini-1.5-pro

```
In [20]: import google.generativeai as genai
import os
from dotenv import load_dotenv

load_dotenv()

# Configure the Gemini API with your API key
genai.configure(api_key=os.environ["GEMINI_API_KEY"])
```

```
In [8]: prompt = f"Question: {random_question['question']}\n\nPlease solve this problem step by step."
```

```
In [10]: # Initialize the model
model = genai.GenerativeModel(model_name="gemini-1.5-pro")

# Generate a response
response = model.generate_content(prompt)

# Display the model's response
print("Model's Response:")
print(response.text)
```

Model's Response:

```
1. **Define variables:**
    * Let 'a' represent the number of Pokemon Alex caught.
    * Let 's' represent the number of Pokemon Stan caught.
    * Let 'w' represent the number of Pokemon Adelwolfe caught.

2. **Set up equations based on the given information:**
    *  $a + s + w = 339$  (They caught 339 Pokemon in total)
    *  $a = s + 5$  (Alex caught 5 more than Stan)
    *  $s = 4w - 13$  (Stan caught 13 less than 4 times Adelwolfe's catch)

3. **Solve for 'w' (Adelwolfe's Pokemon):**
    * Substitute the first two equations into the third to eliminate 'a':
       $(s + 5) + s + w = 339$ 
       $2s + w + 5 = 339$ 
       $2s + w = 334$ 
    * Now we have two equations with 's' and 'w':
       $s = 4w - 13$ 
       $2s + w = 334$ 
    * Substitute the first equation into the second:
       $2(4w - 13) + w = 334$ 
       $8w - 26 + w = 334$ 
       $9w - 26 = 334$ 
       $9w = 360$ 
       $w = 40$ 

4. **Solve for 's' (Stan's Pokemon):**
    * Substitute the value of 'w' back into the equation  $s = 4w - 13$ :
       $s = 4 * 40 - 13$ 
       $s = 160 - 13$ 
       $s = 147$ 

5. **Solve for 'a' (Alex's Pokemon) (Optional – not required by the question but good to check):**
    * Substitute the value of 's' into  $a = s + 5$ :
       $a = 147 + 5$ 
       $a = 152$ 

6. **Check the answer:**
    *  $a + s + w = 152 + 147 + 40 = 339$  (Correct)

**Answer:** Stan caught 147 Pokemon.
```


1. Prompt Engineering

```

In [15]: # General function to generate a solution based on a prompt and a problem
def generate_solution(prompt, problem):
    full_prompt = f"{prompt}\n\nQuestion: {problem}\nLet's think step by step."
    response = model.generate_content(full_prompt)
    return response.text.strip()

# One-shot prompting for numeric answers
def one_shot_prompting_numeric(problem_to_solve):
    example_question = (
        "A store sells notebooks in packs of 12 and markers in packs of 8. "
        "If someone buys 5 packs of notebooks and 3 packs of markers, "
        "how many individual items do they have in total?"
    )
    example_solution = (
        "First, calculate the total number of notebooks: 5 packs * 12 = 60.\n"
        "Next, calculate the total number of markers: 3 packs * 8 = 24.\n"
        "Now, add these together: 60 + 24 = 84.\n"
        "The answer is 84."
    )
    one_shot_prompt = (
        f"Question: {example_question}\nSolution: {example_solution}\n\n"
        f"Now solve this:\n{problem_to_solve}\nSolution:"
    )
    return generate_solution(one_shot_prompt, problem_to_solve)

# Two-shot prompting for numeric answers
def two_shot_prompting_numeric(problem_to_solve):
    example_1_question = (
        "A factory produces 120 gadgets per hour. If it operates 8 hours per day, "
        "how many gadgets does it produce in 5 days?"
    )
    example_1_solution = (
        "First, calculate the daily production: 120 gadgets * 8 hours = 960 gadgets per day.\n"
        "Then, calculate the production over 5 days: 960 * 5 = 4800.\n"
        "The answer is 4800."
    )
    example_2_question = (
        "Sarah has a collection of 90 stamps. She gives 1/3 of them to her friend, "
        "and then buys 30 more. How many stamps does she have now?"
    )
    example_2_solution = (
        "First, calculate how many stamps she gives away: 90 * 1/3 = 30.\n"
        "Subtract this from her original collection: 90 - 30 = 60.\n"
        "Now add the stamps she bought: 60 + 30 = 90.\n"
        "The answer is 90."
    )

```

```

    )
    two_shot_prompt = (
        f"Question: {example_1_question}\nSolution: {example_1_solution}\n\n"
        f"Question: {example_2_question}\nSolution: {example_2_solution}\n\n"
        f"Now solve this:\n{problem_to_solve}\nSolution:"
    )
    return generate_solution(two_shot_prompt, problem_to_solve)

# Two-shot Chain-of-Thought (CoT) prompting
def two_shot_cot_prompting(problem_to_solve):
    example_1_question = (
        "A school buys 4 sets of books, each containing 15 books. The school receives a donation of 20 more books. If each classroom receives an equal share of 5 classrooms, how many books does each classroom get?"
    )
    example_1_cot_solution = (
        "First, calculate the total books purchased: 4 sets * 15 = 60 books.\n"
        "Next, add the donated books: 60 + 20 = 80.\n"
        "Now, divide this among the classrooms: 80 / 5 = 16.\n"
        "Each classroom gets 16 books."
    )
    example_2_question = (
        "A gardener plants 3 rows of 25 flowers each in the morning, and 2 rows of 30 flowers each in the afternoon. In the evening, 10 flowers are eaten by deer. How many flowers are left in the garden?"
    )
    example_2_cot_solution = (
        "First, calculate the flowers planted in the morning: 3 * 25 = 75.\n"
        "Then, calculate the flowers planted in the afternoon: 2 * 30 = 60.\n"
        "Add them to get the total flowers: 75 + 60 = 135.\n"
        "Now, subtract the flowers eaten by deer: 135 - 10 = 125.\n"
        "The answer is 125."
    )
    cot_prompt = (
        f"Question: {example_1_question}\nSolution: {example_1_cot_solution}\n\n"
        f"Question: {example_2_question}\nSolution: {example_2_cot_solution}\n\n"
        f"Now solve this step-by-step:\n{problem_to_solve}\nSolution:"
    )
    return generate_solution(cot_prompt, problem_to_solve)

```

```
In [14]: random_question = gsm8k[random.randint(0, len(gsm8k) - 1)]
# Display the selected question
print("Selected Question:")
print("Question:", random_question['question'])
print("Answer:", random_question['answer'])

print("One-shot Prompting:")
print(one_shot_prompting_numeric(random_question['question']))

print("Two-shot Prompting:")
print(two_shot_prompting_numeric(random_question['question']))

print("Two-shot Chain-of-Thought (CoT) Prompting:")
print(two_shot_cot_prompting(random_question['question']))
```

Selected Question:

Question: Dylan attended a wedding where there were 100 guests in the reception. Each guest brought a plate of 15 cookies. The bride decided to give $\frac{1}{2}$ of the cookies to the church next door as a thank you for helping in the wedding reception. If each person in the church next door got 15 cookies, how many people were in the church next door?

Answer: If each of the 100 guests brought a plate of 15 cookies, then the total number of cookies brought is $100 \times 15 = 1500$ cookies.

If the next church was given $\frac{1}{2}$ the number of cookies, then they got $\frac{1}{2} \times 1500 = 750$ cookies.

If each person in the church next door got 15 cookies, then there were $750 / 15 = 50$ people.

50

One-shot Prompting:

1. **Calculate the total number of cookies:** 100 guests * 15 cookies/guest = 1500 cookies

2. **Calculate the number of cookies given to the church:** 1500 cookies * $(\frac{1}{2})$ = 750 cookies

3. **Calculate the number of people in the church:** 750 cookies / 15 cookies/person = 50 people

Therefore, there were 50 people in the church next door.

Final Answer: The final answer is $\boxed{50}$

Two-shot Prompting:

1. **Calculate the total number of cookies:** 100 guests * 15 cookies/guest = 1500 cookies

2. **Calculate the number of cookies given to the church:** 1500 cookies * $(\frac{1}{2})$ = 750 cookies

3. **Calculate the number of people in the church:** 750 cookies / 15 cookies/person = 50 people

Therefore, there were 50 people in the church next door.

Answer: 50

Two-shot Chain-of-Thought (CoT) Prompting:

1. **Calculate the total number of cookies:** 100 guests * 15 cookies/guest = 1500 cookies

2. **Calculate the number of cookies given to the church:** 1500 cookies * $(\frac{1}{2})$ = 750 cookies

3. **Calculate the number of people in the church:** 750 cookies / 15 cookies/person = 50 people

Therefore, there were 50 people in the church next door.

1. Prompt Refinement

```

In [16]: # Updated function to generate a solution
def generate_solution_refined(prompt, problem):
    refined_prompt = f"{prompt}\n\nQuestion: {problem}\nLet's work this out carefully and verify each step."
    response = model.generate_content(refined_prompt)
    return response.text.strip()

# Refined one-shot prompting
def refined_one_shot_prompting_numeric(problem_to_solve):
    example_question = (
        "A warehouse has 150 boxes. A truck carries away 3 loads of 30 boxes each, "
        "then 20 more boxes are added. How many boxes remain?"
    )
    example_solution = (
        "First, calculate how many boxes the truck carries away:  $3 * 30 = 90$ .\n"
        "Subtract this from the initial count:  $150 - 90 = 60$ .\n"
        "Now, add the extra boxes:  $60 + 20 = 80$ .\n"
        "The answer is 80."
    )
    refined_prompt = (
        f"Question: {example_question}\nSolution: {example_solution}\n\n"
        f"Now solve this carefully and check your answer:\n{problem_to_solve}\nSolution:"
    )
    return generate_solution_refined(refined_prompt, problem_to_solve)

# Refined two-shot prompting
def refined_two_shot_prompting_numeric(problem_to_solve):
    example_1_question = (
        "A train has 120 seats. If it fills 4 carriages with 20 seats each, how many seats remain empty?"
    )
    example_1_solution = (
        "First, calculate the filled seats:  $4 * 20 = 80$ .\n"
        "Subtract this from the total seats:  $120 - 80 = 40$ .\n"
        "The answer is 40."
    )
    example_2_question = (
        "If a bookstore has 200 books and sells 35% of them, how many books are left?"
    )
    example_2_solution = (
        "Calculate 35% of 200:  $200 * 0.35 = 70$ .\n"
        "Subtract this from the total:  $200 - 70 = 130$ .\n"
        "The answer is 130."
    )
    refined_prompt = (
        f"Question: {example_1_question}\nSolution: {example_1_solution}\n\n"
        f"Question: {example_2_question}\nSolution: {example_2_solution}\n\n"
        f"Now solve this carefully, performing each operation:\n{problem_to_solve}\nSolution:"
    )

```

```

    return generate_solution_refined(refined_prompt, problem_to_solve)

# Refined two-shot Chain-of-Thought (CoT) prompting
def refined_two_shot_cot_prompting(problem_to_solve):
    example_1_question = (
        "A farmer has 3 fields. Each field has 20 apple trees. If he p
lants 5 more trees in each field, "
        "how many trees are there in total?"
    )
    example_1_cot_solution = (
        "First, calculate the initial trees: 3 fields * 20 = 60.\n"
        "Now add the extra trees per field: 3 * 5 = 15.\n"
        "Total trees: 60 + 15 = 75.\n"
        "The answer is 75."
    )
    example_2_question = (
        "A hiker walks 3 miles every day for 5 days, then 4 miles each
day for the next 3 days. "
        "How many miles has he walked in total?"
    )
    example_2_cot_solution = (
        "Calculate the distance for the first 5 days: 3 miles * 5 = 1
5.\n"
        "Next, for the following 3 days: 4 miles * 3 = 12.\n"
        "Add them together: 15 + 12 = 27.\n"
        "The answer is 27."
    )
    refined_cot_prompt = (
        f"Question: {example_1_question}\nSolution: {example_1_cot_sol
ution}\n\n"
        f"Question: {example_2_question}\nSolution: {example_2_cot_sol
ution}\n\n"
        f"Now solve this step-by-step, verifying each calculation:\n{p
roblem_to_solve}\nSolution:"
    )
    return generate_solution_refined(refined_cot_prompt, problem_to_so
lve)

```

1. Evaluation

- Testing with original prompts above

```
In [17]: random_question = gsm8k[random.randint(0, len(gsm8k) - 1)]
print("Selected Question:")
print("Question:", random_question['question'])
print("Answer:", random_question['answer'])

print("Refined One-shot Prompting:")
print(refined_one_shot_prompting_numeric(random_question['question']))

print("Refined Two-shot Prompting:")
print(refined_two_shot_prompting_numeric(random_question['question']))

print("Refined Two-shot Chain-of-Thought (CoT) Prompting:")
print(refined_two_shot_cot_prompting(random_question['question']))
```


Selected Question:

Question: Tom went on a two-week-long trip through Europe. In the first 4 days, he traveled 200 kilometers every day, and over the next two days, he totaled only 30% of the distance traveled over the first four days. On the next day, he wasn't traveling at all. During the second week, he made 300 kilometers every day. How many kilometers in total did Tom make during his two-week-long trip?

Answer: In the first four days, Tom made $4 * 200 = 800$ kilometers.

For the next two days, he made only $30/100 * 800 = 240$ kilometers.

During the second week, Tom made 300 kilometers every day, which means $7 * 300 = 2100$ kilometers during the whole week.

During his whole trip, Tom made $800 + 240 + 2100 = 3140$ kilometers.

3140

Refined One-shot Prompting:

1. **First 4 days:** Tom traveled $200 \text{ km/day} * 4 \text{ days} = 800 \text{ km}$
2. **Next 2 days:** He traveled 30% of 800 km = $0.30 * 800 \text{ km} = 240 \text{ km}$
3. **Next day:** 0 km (he didn't travel)
4. **First week total:** $800 \text{ km} + 240 \text{ km} + 0 \text{ km} = 1040 \text{ km}$
5. **Second week:** A week has 7 days, so he traveled $300 \text{ km/day} * 7 \text{ days} = 2100 \text{ km}$
6. **Total trip:** $1040 \text{ km (first week)} + 2100 \text{ km (second week)} = 3140 \text{ km}$

Therefore, Tom traveled a total of 3140 kilometers during his two-week trip.

Final Answer: The final answer is $\boxed{3140}$

Refined Two-shot Prompting:

1. **First 4 days:** Tom traveled $200 \text{ km/day} * 4 \text{ days} = 800 \text{ km}$
2. **Next 2 days:** Tom traveled 30% of 800 km = $0.30 * 800 \text{ km} = 240 \text{ km}$
3. **Following day:** Tom traveled 0 km.
4. **Second week:** A week has 7 days. Tom traveled $300 \text{ km/day} * 7 \text{ days} = 2100 \text{ km}$
5. **Total distance:** $800 \text{ km} + 240 \text{ km} + 0 \text{ km} + 2100 \text{ km} = 3140 \text{ km}$

The answer is $\boxed{3140}$

Refined Two-shot Chain-of-Thought (CoT) Prompting:

Solution:

1. **First 4 days:** Tom traveled $200 \text{ km/day} * 4 \text{ days} = 800 \text{ km}$. ($200 * 4 = 800$ - Correct)
2. **Next 2 days:** He traveled 30% of the distance of the first 4 days. That's $0.30 * 800 \text{ km} = 240 \text{ km}$. ($0.30 * 800 = 240$ - Correct)

3. **Following day:** Tom didn't travel at all, so 0 km.

4. **Second week:** A week has 7 days. He traveled 300 km/day * 7 days = 2100 km. (300 * 7 = 2100 – Correct)

5. **Total distance:** Add up the distances from all periods: 800 km + 240 km + 0 km + 2100 km = 3140 km. (800 + 240 + 0 + 2100 = 3140 – Correct)

The answer is $\boxed{3140}$.

My initial prompts were sufficient for answering all of the random questions I started with, however the answers with my refined prompts are very detailed in their solution walkthroughs.

1. One-Shot Prompting:

- **Original Prompt:** In the original one-shot prompt, the model accurately followed steps to calculate total cookies and distribute them, providing a clear and correct final answer. However, the steps were somewhat basic and didn't encourage deeper validation, which is sufficient for straightforward calculations.
- **Refined Prompt:** The refined one-shot prompt introduced clearer, more structured steps and explicitly encouraged the model to check its calculations. This led to a more detailed breakdown, which was beneficial in multi-part problems, ensuring careful handling of each operation and adding precision.

1. Two-Shot Prompting:

- **Original Prompt:** The original two-shot prompt correctly solved simple calculations, adding a second example for guidance. It correctly computed each step, but similar to the one-shot approach, it did not prompt the model to verify its answer thoroughly.
- **Refined Prompt:** The refined two-shot prompt further improved accuracy by adding explicit instructions for multi-step operations (e.g., calculating percentages or working through specific periods in a week). This additional guidance helped in breaking down complex problems and was particularly useful in cases where multiple steps were required, leading to higher accuracy and a well-organized response.

1. Two-Shot Chain-of-Thought (CoT) Prompting:

- **Original CoT Prompt:** The original CoT prompt demonstrated improved logical reasoning over one-shot and two-shot prompts by breaking down the problem into intermediate steps. However, while it produced correct answers, it did not explicitly validate each calculation step-by-step, which could result in errors for more intricate problems.
- **Refined CoT Prompt:** In the refined CoT prompt, the model was encouraged not only to solve the problem step-by-step but also to verify each calculation along the way. This refinement was particularly effective in managing complex, multi-part problems (e.g., Tom's travel distances over varying days). By prompting the model to confirm each intermediate answer, the refined CoT prompt ensured accuracy across multiple stages, making it ideal for detailed reasoning.

In summary, the refined prompting techniques, especially the two-shot CoT prompt, led to enhanced clarity, accuracy, and reliability in solving complex problems by adding verification steps and specific instructions.

1. The majority of my work, school, and hobby-related experience with LLMs has been with OpenAI models such as GPT-3.5-turbo, GPT-4, and GPT-4o. I have some experience at work with LLama2 7B, but for the most part I have utilized the OpenAI API to access their models. I took this problem as an opportunity to explore Gemini (gemini-1.5-pro) and its API which was pleasantly easy to set up.

I was worried that the prompt engineering skills I've built over the past few years with the GPT models wouldn't translate cleanly to Gemini, but I was still able to successfully craft prompts that delivered correct solutions which I was happy about. I am not sure whether it is just in my head or not, but I do think that the overall feel/tone of the responses, as well as the formatting is different from what I'm used to. Nevertheless, the more concise information and instructions I provided in the prompts, the better the responses. I thought that my refined, more complex responses may be difficult for the model to interpret and extrapolate in order to solve the unseen problems, but it performed well for my small sample of random questions

Problem 4

1. Implementing search tool using SerpAPI

```

In [1]: from langchain.agents import Tool
        from serpapi import GoogleSearch
        import os
        from dotenv import load_dotenv

        load_dotenv()

        class SearchTool(Tool):
            def __init__(self, api_key=None):
                super().__init__(name="search", description="Performs a web search using SerpAPI.", func=self._run)

            def _run(self, query: str) -> str:
                """Run the search query using SerpAPI and return formatted results."""
                params = {
                    "engine": "google",
                    "q": query,
                    "api_key": os.getenv("SERP_API_KEY"),
                }
                search = GoogleSearch(params)
                results = search.get_dict()

                # Format the results
                formatted_results = self.format_results(results)

                # Join results into a single string for easier display in the ReAct agent
                formatted_output = "\n".join(
                    [f"Title: {item['title']}\nLink: {item['link']}\nSnippet: {item['snippet']}\n" for item in formatted_results]
                )

                return formatted_output

            def format_results(self, results):
                """Format search results to return only relevant information."""
                formatted_results = []

                if "organic_results" in results:
                    for result in results["organic_results"]:
                        title = result.get("title", "No title")
                        link = result.get("link", "No link")
                        snippet = result.get("snippet", "No snippet")

                        # Each result is a dictionary with title, link, and snippet for easy access by the ReAct agent
                        formatted_results.append({
                            "title": title,
                            "link": link,
                            "snippet": snippet
                        })
                else:
                    formatted_results.append({"error": "No results found"})

                return formatted_results

```

1. Create comparison tool

```

In [77]: from langchain.tools import Tool
from langchain.prompts import PromptTemplate
from typing import List, Dict
import re

class ComparisonTool(Tool):
    def __init__(self):
        super().__init__(
            name="ComparisonTool",
            description="Compares items based on a specified category",
            func=self._run # Pass _run as the function to execute
        )

    def _run(self, tool_input: str):
        # Parse items and category from the input string
        match = re.search(r'items: \[(.*?)\], category: "\\'(.*?)'"
\''', tool_input)
        if match:
            items = [item.strip() for item in match.group(1).split
(",")]
            category = match.group(2)
        else:
            return f"Error: Input format is incorrect. Expected format
is 'items: [item1, item2, ...], category: \"category_name\"'. Receive
d: {tool_input}"

        # Error handling for invalid inputs
        if not items or len(items) < 2:
            return "Error: Provide at least two items for comparison."
        if not category:
            return "Error: Please specify a category for comparison."

        # Format the items and apply the prompt template
        prompt_template = PromptTemplate(
            input_variables=["items", "category"],
            template="Compare the following items based on {category}.
Provide a summary highlighting key differences and similarities:\n\n{i
tems}"
        )
        formatted_items = "\n".join([f"- {item}" for item in items])
        prompt = prompt_template.format(items=formatted_items, categor
y=category)

        # Comparison logic (simulated response for this example)
        result = self.compare_items(prompt)

        return result

    def compare_items(self, prompt: str) -> str:
        # This function simulates a response from a model using the pr
ompt.

        # Replace with actual model call in a production setting.
        response = f"Comparison based on {prompt.split('based on')[1].
split('.')[0]}:\n"
        response += "Key points:\n1. Item A has ...\n2. Item B differs

```

```
by ... \n3. Similarities include ..."  
    return response
```

1. Implement analysis tool

```

In [76]: from langchain_google_genai import ChatGoogleGenerativeAI
import google.generativeai as genai
from langchain.agents import Tool
import os

class AnalysisTool(Tool):
    def __init__(self, api_key=None):
        super().__init__(name="analyze", description="Analyzes and summarizes search or comparison results.", func=self._run)
        genai.configure(api_key=os.getenv("GEMINI_API_KEY"))

    def _run(self, content):
        # Accepts plain text summaries or lists of snippets
        if not content or len(content) == 0:
            return "Error: No content provided for analysis."

        # If content is a list of dicts, convert to a formatted string
        if isinstance(content, list) and all(isinstance(item, dict) for item in content):
            content = self.format_content(content)
        elif isinstance(content, str):
            # If content is already a string, use it directly
            pass
        else:
            return "Error: Unsupported content format. Please provide a list of dictionaries or a summary string."

        prompt = self.create_prompt(content)

        try:
            model = genai.GenerativeModel(model_name="gemini-1.5-pro")
            response = model.generate_content(prompt)
            return response.text.strip()
        except Exception as e:
            return f"Error in generating analysis: {str(e)}"

    def format_content(self, content):
        """Formats list of dictionaries for analysis."""
        formatted_content = ""
        for item in content:
            title = item.get("title", "No title")
            snippet = item.get("snippet", "No snippet available")
            formatted_content += f>Title: {title}\nSnippet: {snippet}\n\n"
        return formatted_content.strip()

    def create_prompt(self, content):
        """Creates a structured prompt for analysis."""
        prompt = (
            "Analyze the following information and provide a concise summary of key points:\n\n"
            f"{content}\n\nSummary:"
        )
        return prompt

```


1. Integrate tools

```
In [78]: from langchain.agents import create_react_agent, AgentExecutor
         from langchain.prompts import PromptTemplate
```

```
In [79]: # Initialize tools
         search_tool = SearchTool()
         comparison_tool = ComparisonTool()
         analysis_tool = AnalysisTool()

         tools = [search_tool, comparison_tool, analysis_tool]
```

```
In [80]: template = '''Answer the following questions as best you can. You have
         access to the following tools:

         {tools}

         Use the following format:

         Question: the input question you must answer
         Thought: you should always think about what to do
         Action: the action to take, should be one of [{tool_names}]
         Action Input: the input to the action
         Observation: the result of the action
         ... (this Thought/Action/Action Input/Observation can repeat N times)
         Thought: I now know the final answer
         Final Answer: the final answer to the original input question

         Begin!

         Question: {input}
         Thought:{agent_scratchpad}'''

         prompt_template = PromptTemplate.from_template(template)
```

```
In [81]: # Initialize the ReAct agent
         agent = create_react_agent(
             llm = ChatGoogleGenerativeAI(model="gemini-1.5-pro"),
             tools=tools,
             prompt=prompt_template
         )

         # Create an AgentExecutor to run the agent
         agent_executor = AgentExecutor(agent=agent, tools=tools, verbose=True,
             handle_parsing_errors=True)
```

```
In [83]: query = "Compare the camera quality of the iPhone 14, Samsung Galaxy S  
21, and Google Pixel 6."  
response = agent_executor.invoke({"input": query})  
  
print("Agent's Response:")  
print(response)
```

> Entering new AgentExecutor chain...

Question: Compare the camera quality of the iPhone 14, Samsung Galaxy S21, and Google Pixel 6.

Thought: I need to gather information about the camera specs and reviews for each phone. I can use search for this.

Action: search

Action Input: "iPhone 14 camera review" Title: Is the Apple iPhone 14 Pro a Good Camera in 2024?

Link: <https://fstoppers.com/mobile/apple-iphone-14-pro-good-camera-2024-review-653471>

Snippet: The improved Lidar technology allows for much more precise portrait captures. As well as that, the iPhone is remarkably good at capturing and ...

Title: iPhone 14 Pro Camera Review: Scotland

Link: <https://www.austinmann.com/trek/iphone-14-pro-camera-review-scotland>

Snippet: The iPhone 14 Pro introduces a massive resolution jump to 48MP, quadrupling the iPhone 13 Pro's sensor.

Title: Apple iPhone 14 Camera test

Link: <https://www.dxomark.com/apple-iphone-14-camera-test/>

Snippet: Pros · Good exposure and nice color · Fast and accurate autofocus · Realistic bokeh effect in portrait mode · Preview image close to capture ...

Title: iPhone 14 Pro Camera Review: A Small Step, A Huge Leap

Link: https://www.reddit.com/r/apple/comments/yjj355/iphone_14_pro_camera_review_a_small_step_a_huge/

Snippet: The new iPhone 14 pro camera is amazing. The biggest flaw to me has been when I attempted to take an up close shot. Usually it would be a ...

Title: I figured out why I don't love the iPhone 14 Pro's camera

Link: <https://www.digitaltrends.com/mobile/i-dont-love-the-iphone-14-pro-camera-and-i-know-why/>

Snippet: The iPhone 14 Pro takes photos with such strong contrast levels and such poor exposure management that many images are too shadowy and dark for ...

Title: Poor quality photos on iPhone 14 pro max

Link: <https://discussions.apple.com/thread/254569049>

Snippet: The iPhone 14 Pro camera desaturates the colours, overly colour corrects to the point it looks horrible, and when used in apps like instagram, the camera is ...

Title: Apple iPhone 14 Pro Review: The Only Camera You ...

Link: <https://petapixel.com/2022/10/06/apple-iphone-14-pro-review-the-only-camera-you-really-need/>

Snippet: The biggest change to the iPhone 14 Pro's camera is the addition of a new 48-megapixel quad-pixel main sensor that Apple says enables better ...

Title: Apple iPhone 14 review: Camera, photo and video quality

Link: https://www.gsmarena.com/apple_iphone_14-review-2481p5.php

Snippet: The main camera on the iPhone 14 captures very clean and detailed

iled low-light shots. Noise is controlled very well and so are light sources. There ...

Thought: I need to gather information about the camera specs and reviews for each phone. I can use search for this.

Action: search

Action Input: "iPhone 14 camera review"Title: Is the Apple iPhone 14 Pro a Good Camera in 2024?

Link: <https://fstoppers.com/mobile/apple-iphone-14-pro-good-camera-2024-review-653471>

Snippet: The improved Lidar technology allows for much more precise portrait captures. As well as that, the iPhone is remarkably good at capturing and ...

Title: iPhone 14 Pro Camera Review: Scotland

Link: <https://www.austinmann.com/trek/iphone-14-pro-camera-review-scotland>

Snippet: The iPhone 14 Pro introduces a massive resolution jump to 48MP, quadrupling the iPhone 13 Pro's sensor.

Title: Apple iPhone 14 Camera test

Link: <https://www.dxomark.com/apple-iphone-14-camera-test/>

Snippet: Pros · Good exposure and nice color · Fast and accurate autofocus · Realistic bokeh effect in portrait mode · Preview image close to capture ...

Title: iPhone 14 Pro Camera Review: A Small Step, A Huge Leap

Link: https://www.reddit.com/r/apple/comments/yjj355/iphone_14_pro_camera_review_a_small_step_a_huge/

Snippet: The new iPhone 14 pro camera is amazing. The biggest flaw to me has been when I attempted to take an up close shot. Usually it would be a ...

Title: I figured out why I don't love the iPhone 14 Pro's camera

Link: <https://www.digitaltrends.com/mobile/i-dont-love-the-iphone-14-pro-camera-and-i-know-why/>

Snippet: The iPhone 14 Pro takes photos with such strong contrast levels and such poor exposure management that many images are too shadowy and dark for ...

Title: Poor quality photos on iPhone 14 pro max

Link: <https://discussions.apple.com/thread/254569049>

Snippet: The iPhone 14 Pro camera desaturates the colours, overly colour corrects to the point it looks horrible, and when used in apps like instagram, the camera is ...

Title: Apple iPhone 14 Pro Review: The Only Camera You ...

Link: <https://petapixel.com/2022/10/06/apple-iphone-14-pro-review-the-only-camera-you-really-need/>

Snippet: The biggest change to the iPhone 14 Pro's camera is the addition of a new 48-megapixel quad-pixel main sensor that Apple says enables better ...

Title: Apple iPhone 14 review: Camera, photo and video quality

Link: https://www.gsmarena.com/apple_iphone_14-review-2481p5.php

Snippet: The main camera on the iPhone 14 captures very clean and detailed low-light shots. Noise is controlled very well and so are light sources. There ...

Thought: I need to gather information about the camera specs and reviews for each phone. I can use search for this.

Action: search

Action Input: "Samsung Galaxy S21 camera review"Title: Samsung Galaxy S21 5G (Exynos) Camera review

Link: <https://www.dxomark.com/samsung-galaxy-s21-5g-exynos-camera-review-s-series-base-model/>

Snippet: Pros · Accurate stills exposure in all conditions · Neutral white balance and natural skin tones · Wide depth of field for group photos ...

Title: S21 5G camera quality : r/GalaxyS21

Link: https://www.reddit.com/r/GalaxyS21/comments/s9n1ld/s21_5g_camera_quality/

Snippet: I played around with the camera today, and you're right, it's better in almost every use case except in lower light with not perfectly still ...

Title: Samsung S21 Ultra Camera Review – A Photographer's ...

Link: <https://robinwong.blogspot.com/2021/02/samsung-s21-ultra-camera-review.html>

Snippet: The wide angle camera of Samsung S21 Ultra is generally very good, under good light, it renders sharp, pleasing images. The fine details and ...

Title: Samsung Galaxy S21 5G review: Camera quality

Link: https://www.gsmarena.com/samsung_galaxy_s21-review-2218p5.php

Snippet: So yes, the S21 takes really sharp selfies, perhaps excessively so. Dynamic range is nicely wide, and backlit scenes aren't an issue either.

Title: S21 Bad Camera Quality

Link: <https://eu.community.samsung.com/t5/other-galaxy-s-series/s21-bad-camera-quality/td-p/3711489>

Snippet: Just recently got the S21 and I'm so disappointed with the camera. The photos it takes are grainy, blurry, low quality and look like something a phone in 2008 ...

Title: Samsung Galaxy S21 Smartphone Review – Performance

Link: <https://www.ephotozine.com/article/samsung-galaxy-s21-smartphone-review--35404/performance>

Snippet: Sample Photos – Image quality is impressive with sharp detail and good exposure levels. Colours are, generally, also accurate but the Galaxy S21 does have a ...

Title: Samsung Galaxy S21 5G (Snapdragon) Camera review

Link: <https://www.dxomark.com/samsung-galaxy-s21-5g-snapdragon-camera-review-a-slight-qualcomm-advantage/>

Snippet: Pros · Fairly neutral white balance and accurate color rendering in stills and video · Stills exposure accurate in bright to moderate light ...

Title: Review: Samsung Galaxy S21 and S21 Ultra – WIRED

Link: <https://www.wired.com/review/samsung-galaxy-s21-ultra/>

Snippet: Fantastic displays and speedy performance. Excellent cameras for photo and video, especially on the top-of-the-line Ultra. Great build quality.

Thought: I need to gather information about the camera specs and reviews for each phone. I can use search for this.

Action: search

Action Input: "Google Pixel 6 camera review"Title: Google Pixel 6 Camera Review – Can It Be A ...

Link: <https://dougashphotography.com/pixel-6-camera-review/>

Snippet: The Google Pixel 6 camera is amazing! Its low-light ability is epic. Even without using the night shot mode.

Title: Google Pixel 6 Camera test: An outstanding performer in its ...

Link: <https://www.dxomark.com/google-pixel-6-camera-review-an-outstanding-performer-in-its-segment/>

Snippet: Pros · Good detail in bright light and indoors · Nice and accurate color · Natural detail and good shadow detail · Fast and consistent autofocus ...

Title: Review after using the Pixel 6 for a few weeks : r/Pixel6

Link: https://www.reddit.com/r/Pixel6/comments/uvmb9c/review_after_using_the_pixel_6_for_a_few_weeks/

Snippet: So aside from the audio issues when recording video, the camera is excellent. Pictures are absolutely perfect and Google photos handles showing ...

Title: Google Pixel 6 review – best phone camera at the price?

Link: <https://amateurphotographer.com/review/google-pixel-6-review-the-best-camera-for-under-600/>

Snippet: The main camera offers up to 7x super-resolution zoom, combining multiple images to produce results with more detail than a standard digital zoom would. However ...

Title: Why Pixel 6 cameras are better than reviews say

Link: <https://www.androidcentral.com/why-pixel-6-cameras-are-better-reviewers-say>

Snippet: The Google Pixel 6 is capable of amazing shots. It's just not amazing every single time. And it's all absolutely right.

Title: Google Pixel 6 Smartphone Review – Performance

Link: <https://www.ephotozine.com/article/google-pixel-6-smartphone-review-35825/performance>

Snippet: The image quality from the Google Pixel 6 is superb. The true-to-life colours are exceptional as is dynamic range and detail/sharpness.

Title: Google Pixel 6 review: Camera: Hardware, app, photo quality

Link: https://m.gsmarena.com/google_pixel_6-review-2350p5.php

Snippet: You get sharp and detailed 8MP images with minimal noise. Dynamic range is excellent, and even scenes with strong backlight will be rendered ...

Title: Google Pixel 6 review: This phone is everything I wanted

Link: <https://www.cnet.com/tech/mobile/google-pixel-6-review/>

Snippet: The Pro has a wider selfie camera that can record 4K. Selfies photos from the Pixel 6 are good but in medium and low light start to look noisy.

Thought: I've gathered information on all three cameras. Now I can use the ComparisonTool to compare them based on camera quality.

Action: ComparisonTool

Action Input: Compare the camera quality of the iPhone 14, Samsung Galaxy S21, and Google Pixel 6. Consider factors such as low light performance, detail, color accuracy, and overall image quality.
Error: Input format is incorrect. Expected format is 'items: [item1, item2, ...], category: "category_name"'. Received: Compare the camera quality of the iPhone 14, Samsung Galaxy S21, and Google Pixel 6. Consider factors such as low light performance, detail, color accuracy, and overall image quality.
Thought: The ComparisonTool requires a specific input format. I need to reformat my input.

Action: ComparisonTool

Action Input: items: [iPhone 14, Samsung Galaxy S21, Google Pixel 6], category: "camera quality"
Error: Please specify a category for comparison.
Thought: I made a mistake in the last input. "camera quality" should be the category.

Action: ComparisonTool

Action Input: items: [iPhone 14, Samsung Galaxy S21, Google Pixel 6], category: "camera quality"
Error: Please specify a category for comparison.
Thought: I keep getting an error, and I'm not sure why. I'll try to be more explicit about the comparison criteria in the category.

Action: ComparisonTool

Action Input: items: [iPhone 14, Samsung Galaxy S21, Google Pixel 6], category: "camera quality: low light performance, detail, color accuracy, overall image quality"
Error: Please specify a category for comparison.
Thought: I'm still getting an error with the ComparisonTool. I'll try analyzing the search results and summarizing the camera quality for each phone, then compare them manually.

Action: analyze

Action Input: The iPhone 14 Pro's new 48MP sensor improves detail and low-light performance. Some reviews mention issues with contrast and exposure. The iPhone 14 (non-Pro) has a good main camera with well-controlled noise in low light. The Samsung Galaxy S21 generally captures sharp images with accurate colors, but some users report grainy or blurry photos in low light. The Google Pixel 6 is praised for its excellent low-light capabilities, natural detail, and accurate colors, although consistency can be an issue.

iPhone 14 Pro: High detail 48MP sensor, improved low light but some contrast/exposure issues. iPhone 14: Good low-light performance with controlled noise. Galaxy S21: Sharp images, accurate colors, but potential low-light struggles. Pixel 6: Excellent low light, natural detail, accurate colors, but occasional inconsistency.
Thought: I now have a summarized comparison of the camera quality for each phone. I can use this information to provide a final answer.

Final Answer: All three phones have strong cameras, but their strengths differ. The iPhone 14 Pro offers the highest detail due to its 48MP sensor and improved low-light performance, but some reviewers have noted issues with contrast and exposure. The standard iPhone 14 also performs well in low light. The Samsung Galaxy S21 excels in sharp images and color accuracy, but its low-light performance might be less consistent. The Google Pixel 6 is a strong contender, especially in low light, with natural detail and accurate colors, but its performance isn't always consistent. The best choice depends on individual priorities. If detail is paramount, the iPhone 14 Pro is a good choice. For reliable color accuracy and sharpness, the Galaxy S21 is a good option. For the best low-light performance, the Pixel 6 is likely the best choice.

> Finished chain.

Agent's Response:

```
{'input': 'Compare the camera quality of the iPhone 14, Samsung Galaxy S21, and Google Pixel 6.', 'output': "All three phones have strong cameras, but their strengths differ. The iPhone 14 Pro offers the highest detail due to its 48MP sensor and improved low-light performance, but some reviewers have noted issues with contrast and exposure. The standard iPhone 14 also performs well in low light. The Samsung Galaxy S21 excels in sharp images and color accuracy, but its low-light performance might be less consistent. The Google Pixel 6 is a strong contender, especially in low light, with natural detail and accurate colors, but its performance isn't always consistent. The best choice depends on individual priorities. If detail is paramount, the iPhone 14 Pro is a good choice. For reliable color accuracy and sharpness, the Galaxy S21 is a good option. For the best low-light performance, the Pixel 6 is likely the best choice."}
```

1. Implement streamlit UI

Code for UI and backend is in agent_tools.py and react_agent_app.py

