What is Transfer Learning?

Transfer learning is a technique used in machine learning, deep learning in which a model already trained on a huge dataset for a specific task is used as a starting point to train a new / same model to adapt for another dataset/task.

Task-1:

Take a VGG-16 model pre-trained on imagenet(for classification task) and adapt it for the new dataset i.e, CIFAR-10 Dataset for classification task.

Steps for Task-1:

1. Import Libraries:

```
# Imports
import torch
import torchvision
import torch.nn as nn  # All neural network modules, nn.Linear, nn.Conv2d, BatchNorm, Loss functions
import torch.optim as optim  # For all Optimization algorithms, SGD, Adam, etc.
import torch.nn.functional as F  # All functions that don't have any parameters like ReLU
from torch.utils.data import (DataLoader)  # Gives easier dataset managment and creates mini batches
import torchvision.datasets as datasets  # Has standard datasets we can import in a nice way
import torchvision.transforms as transforms  # Transformations we can perform on our dataset
```

2. Load the pre-trained model:

```
# Load pretrain model & modify it

model = torchvision.models.vgg16(pretrained=True)
print(model)
```

The VGG-16 model is as follows:

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
```

```
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

We will only modify the **model.classifier** to adapt the model for the CIFAR-10 Dataset.

3. Modify the model:

```
# If you want to do finetuning then set requires_grad = False
# Remove these two lines if you want to train entire model,
# and only want to load the pretrain weights.
for param in model.parameters():
    param.requires_grad = False

#model.avgpool = Identity()
model.classifier = nn.Sequential(
                   nn. Linear(in_features=25088, out_features=4096, bias=True),
                   nn. ReLU(inplace=True),
                   nn.Dropout(p=0.5, inplace=False),
                   nn.Linear(in_features=4096, out_features=4096, bias=True),
                   nn.ReLU(inplace=True),
                   nn.Dropout(p=0.5, inplace=False),
                   nn.Linear(in_features=4096, out_features=1000, bias=True),
                   nn.ReLU(inplace=True),
                   nn.Dropout(p=0.5, inplace=False),
                   nn.Linear(in_features=1000, out_features=10, bias=False))

print(model)
```

4. Check whether the model's input - output dimensions are compatible:

```
x = torch.rand(16,3,224,224)
print(model(x).shape)
```

It should return :

```
torch.Size([16, 10])
```

5. Set device and model hyperparameters:

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

# Hyperparameters
num_classes = 10
learning_rate = 1e-3
batch_size = 1024
num_epochs = 10
```

5. Load CIFAR10 Dataset:

```
# Load Data
train_dataset = datasets.CIFAR10(
    root="dataset/", train=True, transform=transforms.ToTensor(), download=True
)
train_loader = DataLoader(dataset=train_dataset, batch_size=batch_size, shuffle=True)
```

6. Write criterion i.e, loss function and optimizer:

```
# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
```

7. Train the model:

```
# Train Network
model.to(device)
for epoch in range(num_epochs):
    losses = []

    for batch_idx, (data, targets) in enumerate(train_loader):
        # Get data to cuda if possible
        data = data.to(device=device)
        targets = targets.to(device=device)

        # forward
        scores = model(data)
        loss = criterion(scores, targets)

        losses.append(loss.item())
        # backward
        optimizer.zero_grad()
        loss.backward()

        # gradient descent or adam step
        optimizer.step()

    print(f"Cost at epoch {epoch} is {sum(losses)/len(losses):.5f}")
```

Model should start training as follows:

```
Cost at epoch 0 is 1.85966
Cost at epoch 1 is 1.26044
Cost at epoch 2 is 1.18923
Cost at epoch 3 is 1.14436
Cost at epoch 4 is 1.11312
Cost at epoch 5 is 1.08770
Cost at epoch 6 is 1.06205
Cost at epoch 7 is 1.04475
Cost at epoch 8 is 1.01858
Cost at epoch 9 is 1.00933
```

8. Test the model performance on training and testing dataset:

```
# Check accuracy on training & test to see how good our model


def check_accuracy(loader, model):
    if loader.dataset.train:
        print("Checking accuracy on training data")
    else:
        print("Checking accuracy on test data")

    num_correct = 0
    num_samples = 0
    model.eval()

    with torch.no_grad():
        for x, y in loader:
            x = x.to(device=device)
            y = y.to(device=device)

            scores = model(x)
            _, predictions = scores.max(1)
            num_correct += (predictions == y).sum()
            num_samples += predictions.size(0)

        print(
            f"Got {num_correct} / {num_samples} with accuracy {float(num_correct)/float(num_samples)*100:.2f}"
        )

    model.train()
```

```
check_accuracy(train_loader, model)
```

Accuracy should be printed as follows:

```
Checking accuracy on training data
Got 35334 / 50000 with accuracy 70.67
```

Task-2:

Take a VGG-16 model pre-trained on imagenet(for classification task) and adapt it for the new dataset i.e, cat-dog Dataset for object detection task.

Steps to perform task-2:

1. **Download cat-dog object detection dataset**

2. **Import Libraries**

```
import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms
import torch.optim as optim
import torchvision.transforms.functional as FT
from torch.utils.data import DataLoader
from tqdm import tqdm
#from tqdm.auto import tqdm
import pandas as pd
import os
import PIL
import skimage
from skimage import io
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import matplotlib.patches as patches
seed = 123
import cv2
torch.manual_seed(seed)
from collections import Counter
from time import sleep
import random
import torchvision.datasets as datasets  # Has standard datasets we can import in a nice way
from google.colab.patches import cv2_imshow
```

3. **Write LoadData class**
4. **Write function for IOU and convert_boxes_wrt_image**
5. **Write loss function for YOLOv1**
6. **Write training function i.e, train_fn**
7. **Write Compose class for image transformation**
8. **Write functions: tensor_to_boxes and non_max_suppression**
9. **Write functions: get_bboxes and mean_average_precision**

All above steps are the same as the YOLOv1 from scratch done previously.

10. Load pretrained VGG-16 model:

```
# Load pretrain model & modify it

model = torchvision.models.vgg16(pretrained=True)
print(model)
```

VGG-16 model is as follows:

We will modify the **model.classifier** to adapt it for the object detection task.

11. Modify the model:

```
# If you want to do finetuning then set requires_grad = False
# Remove these two lines if you want to train entire model,
# and only want to load the pretrain weights.
for param in model.parameters():
    param.requires_grad = False

#model.avgpool = Identity()
model.classifier = nn.Sequential(
                   nn. Linear(in_features=25088, out_features=4096, bias=True),
                   nn. ReLU(inplace=True),
                   nn.Dropout(p=0.5, inplace=False),
                   nn.Linear(in_features=4096, out_features=4096, bias=True),
                   nn.ReLU(inplace=True),
                   nn.Dropout(p=0.5, inplace=False),
                   nn.Linear(in_features=4096, out_features=1000, bias=True),
                   nn.ReLU(inplace=True),
                   nn.Dropout(p=0.5, inplace=False),
                   nn.Linear(in_features=1000, out_features=588, bias=False),
                   nn.Sigmoid() )

print(model)
```

12. Check whether the model's consecutive inputs-outputs are compatible:

```
x = torch.randn((2,3,224,224))
print(model(x).shape)
```

It should print :

```
torch.Size([2, 588])
```

13. Set device and hyperparameters:

```
LEARNING_RATE = 2e-5
DEVICE = "cuda" if torch.cuda.is_available else "cpu"
BATCH_SIZE = 16  #16 # 64 in original paper but resource exhausted error otherwise.
WEIGHT_DECAY = 0
EPOCHS = 20
NUM_WORKERS = 2
PIN_MEMORY = True
LOAD_MODEL = False
LOAD_MODEL_FILE = "model.pth"
```

14. Perform training:

```
model.to(DEVICE)
optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE, weight_decay=WEIGHT_DECAY)

scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer=optimizer, factor=0.1, patience=3, mode='max', verbose=True)
loss_fn = YOLOLoss()

if LOAD_MODEL:
    load_checkpoint(torch.load(LOAD_MODEL_FILE), model, optimizer)

train_dataset = LoadData(file_dir= '/content/train',transform=transform)

test_dataset = LoadData(file_dir='/content/test',transform=transform)

train_loader = DataLoader(
        dataset=train_dataset,
        batch_size=BATCH_SIZE,
        shuffle=True,
        drop_last=False,
    )

test_loader = DataLoader(
        dataset=test_dataset,
        batch_size=BATCH_SIZE,
        shuffle=True,
        drop_last=False,
    )

for epoch in range(EPOCHS):
        #print(f"Epoch-{epoch}")
    train_fn(train_loader, model, optimizer, loss_fn, epoch)

    pred_boxes, target_boxes = get_bboxes(
            train_loader, model, iou_threshold=0.5, threshold=0.4, device=DEVICE )

    mean_avg_prec = mean_average_precision(
            pred_boxes, target_boxes, iou_threshold=0.5)
    print(f"Train mAP: {mean_avg_prec}")

    scheduler.step(mean_avg_prec)

checkpoint = {
            "state_dict": model.state_dict(),
            "optimizer": optimizer.state_dict(),
    }
save_checkpoint(checkpoint, filename=LOAD_MODEL_FILE)
```

It should start training the model:

```
Epoch 0: 100%|██████████| 161/161 [01:20<00:00,  2.00batch/s]
Mean loss was 75.95041142339292
Epoch 1:   0%|          | 0/161 [00:00<?, ?batch/s]Train mAP: 0.0
Epoch 1: 100%|██████████| 161/161 [01:25<00:00,  1.88batch/s]
Mean loss was 43.05265763087302
Epoch 2:   0%|          | 0/161 [00:00<?, ?batch/s]Train mAP: 0.0
Epoch 2: 100%|██████████| 161/161 [01:25<00:00,  1.89batch/s]
Mean loss was 39.000006800112516
Epoch 3:   0%|          | 0/161 [00:00<?, ?batch/s]Train mAP: 0.001104999566450715
Epoch 3: 100%|██████████| 161/161 [01:25<00:00,  1.89batch/s]
Mean loss was 35.53345983517096
Epoch 4:   0%|          | 0/161 [00:00<?, ?batch/s]Train mAP: 0.1149347573518753
Epoch 4: 100%|██████████| 161/161 [01:25<00:00,  1.88batch/s]
Mean loss was 32.37669768244584
Epoch 5:   0%|          | 0/161 [00:00<?, ?batch/s]Train mAP: 0.2521315813064575
Epoch 5: 100%|██████████| 161/161 [01:25<00:00,  1.88batch/s]
Mean loss was 29.302443225931675
Epoch 6:   0%|          | 0/161 [00:00<?, ?batch/s]Train mAP: 0.39158064126968384
Epoch 6: 100%|██████████| 161/161 [01:25<00:00,  1.88batch/s]
Mean loss was 26.71432886064423
Epoch 7:   0%|          | 0/161 [00:00<?, ?batch/s]Train mAP: 0.45775383710861206
Epoch 7: 100%|██████████| 161/161 [01:25<00:00,  1.87batch/s]
Mean loss was 24.87837935974879
Epoch 8:   0%|          | 0/161 [00:00<?, ?batch/s]Train mAP: 0.5211357474327087
Epoch 8: 100%|██████████| 161/161 [01:25<00:00,  1.89batch/s]
Mean loss was 23.055909310808833
Epoch 9:   0%|          | 0/161 [00:00<?, ?batch/s]Train mAP: 0.5980434417724609
Epoch 9: 100%|██████████| 161/161 [01:25<00:00,  1.89batch/s]
Mean loss was 21.662961675513603
```

15. Evaluate the model on the test dataset:

```
""" Evaluate the model on test dataset"""


LOAD_MODEL_FILE = '/content/model.pth'
#LOAD_MODEL_FILE = '/content/drive/MyDrive/SpanIdea/YOLO_from_scratch/weights/model_22.7_mAP.pth'

optimizer = optim.Adam(
       model.parameters(), lr=LEARNING_RATE, weight_decay=WEIGHT_DECAY
   )
load_checkpoint(torch.load(LOAD_MODEL_FILE), model, optimizer)

model.eval()
#Get mAP on test data:

test_dataset = LoadData(file_dir='/content/test',transform=transform)

test_loader = DataLoader(dataset=test_dataset, batch_size=BATCH_SIZE, shuffle=False, drop_last=False)

pred_boxes, target_boxes = get_bboxes(test_loader, model, iou_threshold=0.5, threshold=0.4, device=DEVICE ,
                                      train = False)

mean_avg_prec = mean_average_precision(pred_boxes, target_boxes, iou_threshold=0.5)
print(f"Test mAP: {mean_avg_prec}")
```

Test results:

```
=> Loading checkpoint
Test mAP: 0.7430607080459595
```

16. Visualize the detection results

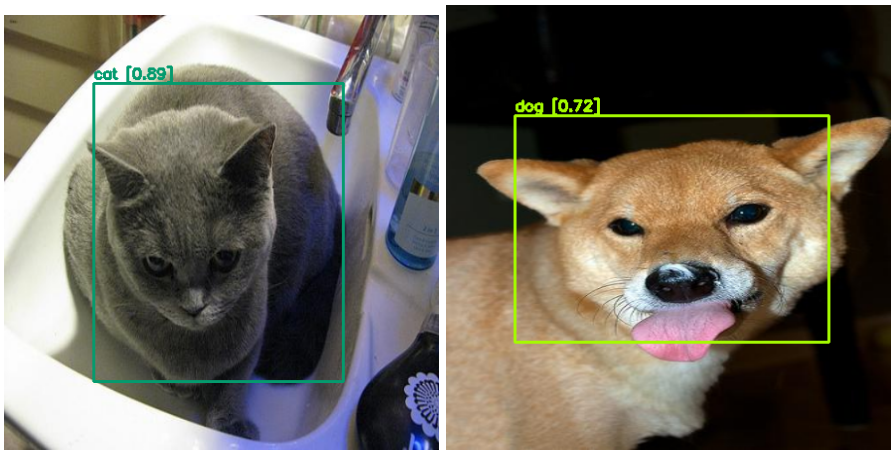
Results:

Training mAP: `0.817078709602356` | Test mAP: `0.7430607080459595`

`Hence, we can say that our model generalizes better when we use a pre-trained model.`

Example of Detection:

Correct predictions:



Incorrect prediction: