Transfer Learning for Classification

Please save a copy of the notebook in your local drive! Otherwise you will lose your progress.

Introduction & a Bit of History

Over the course of CS189, you have trained machine learning models for specific applications, whether it is classification or regression. Although this has been effective, modern machine learning researchers use a considerably different pipeline for training models.

What people have been doing instead is to train a massive model for a very broad objective, and then finetune the model on a smaller dataset. This approach has given us a lot of great-performing models and agents, the most notable being that of GPT models (there are also more models that does the same job, BERT, DINO, masked autoencoders are the ones that immediately came to mind, but unfortunately there are no equivalent in robotics), as they relied on internet scale pretraining (with GPT specifically, people use what's called next-token prediction as the objective). You don't need to know exactly how this works in this class, but this is drastically different than what we have done by fitting a model to exactly one dataset.

Of course, we do not have the computational resources to retrain any GPT3+ models, but we can try a smaller problem with transfer learning! Here is a little story behind the origin of transfer learning:

Back in 2013, neural-net based image recognition methods such as AlexNet has already swept through the ML community by storm, and people are trying to tackle the next problem in computer vision, which is object segmentation (using bounding boxes). The problem is that the corresponding dataset, Pascal-VOC, is very small for neural networks (only 1500 training images and 20,000 data entries).

So what did people do? Researchers have found out is that instead of training a neural network from scratch, using weights from networks trained on ImageNet classification as the starting point actually helped to achieve very good tracking accuracy on these tasks (if you are so inclined, feel free to read R-CNN to see what they have done)! In fact, this method worked so well that it not only inspired the idea of "pretraining" across the machine learning community, but also sparked a funny bet between two UC Berkeley professors.

In previous semesters, we actually didn't want you to do transfer learning for CIFAR-10, but with how much the machine learning world has changed, we are changing the focus specifically on transfer learning to get you more familiar with these paradigms and hope you can have successes working with these large models!

Transfer Learning for CIFAR-10

In this question, we will explore how to effectively use transfer learning to learn a new task with a *pretrained* network. In essence, the idea of pretraining is that a network has been trained on another objective, and that the weights are something more desirable than random initialization. Transfer learning takes advantage of that and aims to make training faster compared to that of training from scratch.

```
# utils for data loading

import torch
import torch.nn as nn
import torch.nn.functional as F
from torchvision import datasets
import torchvision.transforms as transforms
from torch.utils.data import DataLoader, random_split
from torchvision import models

import time
import numpy as np
import pandas as pd

import os
import time

import random
import matplotlib.pyplot as plt

if torch.cuda.is_available():
    torch.backends.cudnn.deterministic = True

%reload_ext autoreload
```

```
%autoreload 2
%matplotlib inline
```

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

```
seed = 7

np.random.seed(seed)
torch.manual_seed(seed)
torch.cuda.manual_seed(seed)
```

First, let's start with making the necessary transformations of our training data. CIFAR-10 is a collection of 60,000 32x32x3 images divided into 10 classes. If you want more information, please refer to this link by Alex Krizhevsky et al.

However, most of the models readily available are trained on ImageNet, which is 224x224x3. Although one can hack the weights of a neural network and design a new network on top of it, it is much easier to simply scale up the images by a factor of 7 (224/32) and feed the network the magnified images. Let's do that instead!

```
## transforms to match that of the transforms in ImageNet
transform = transforms.Compose([
    transforms.Resize((224, 224)), # Resize to 224x224 (ImageNet size)
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])

## download training dataset
train_dataset = datasets.CIFAR10(root='./data', train=True,
                                 transform=transform, download=True)

test_dataset = datasets.CIFAR10(root='./data', train=False,
                                transform=transform, download=True)
```

```
100%|██████████| 170M/170M [00:04<00:00, 40.8MB/s]
```

Let's also visualize some images within CIFAR-10 Dataset!

```
import torchvision

viz_transform = transforms.Compose([
    transforms.Resize((224, 224)), # Resize to 224x224 (height x width)
    transforms.ToTensor(),
])

viz_dataset = datasets.CIFAR10(root='./data', train=True,
                               transform=viz_transform, download=True)

images = [viz_dataset[i][0] for i in range(9)]
plt.imshow(torchvision.utils.make_grid(torch.stack(images), nrow=3, padding=5).numpy().transpose((1, 2, 0)))
```

```
<matplotlib.image.AxesImage at 0x7f5cd5d39150>
```

Now go to *torchvision.models* module, and pick one of the following models that you would like.

- VGG Networks (all VGG networks are fine)
- AlexNet
- ResNet-18, -34, or -50

A more expressive model, such as using vision transformers, will not be allowed (they also have hundreds of millions of parameters, which is not friendly for your colab compute units). All model weights must directly come from *torchvision.models* module.

In order to perform transfer learning, you should do the following steps:

1. Load a pretrained network on another dataset.
2. Freeze most of the layers of the pretrained network and discard a part of the network not relevant to your model.
3. Append a new MLP at the end of the network, and allow gradients to pass through the new MLP (and other non-frozen parts of the network).
4. Train the new network.

Although we would like you to explore transfer learning, here are some rules you must follow:

1. You may not change the model architecture other than the final linear layers.
2. You must freeze all of the weights in the convolutional layers.
3. You may not have more than 3 million trainable parameters.

```
class TransferCIFAR10(nn.Module):
  def __init__(self, *args, **kwargs):
    super(TransferCIFAR10, self).__init__()


    #picked Alexanet
    self.model = models.alexnet(pretrained=True)

    for parameter in self.model.features.parameters():
      parameter.requires_grad = False

    self.model.classifier = nn.Sequential(
      nn.Dropout(0.5),
      nn.Linear(256 * 6 * 6, 4096),
      nn.ReLU(inplace=True),
      nn.Dropout(0.5),
      nn.Linear(4096, 1024),
      nn.ReLU(inplace=True),
      nn.Linear(1024, 10)
    )


  def forward(self, x):
    return self.model(x)
```

If you want to check how many trainable parameters are within your model, run this script here:

```
def num_trainable_params(model: nn.Module):
  return sum(p.numel() for p in model.parameters() if p.requires_grad)


# training script
################################## YOUR CODE HERE ###################################
import tqdm
import torch.optim as optim

model = TransferCIFAR10()
model = model.to(device)

train_size = int(0.8*len(train_dataset))
val_size = len(train_dataset) - train_size
train_data, val_data = random_split(train_dataset, [train_size, val_size])

train_loaded = DataLoader(train_data, batch_size=64, shuffle=True)
val_loaded = DataLoader(val_data, batch_size=64, shuffle=True)

criteria = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

epochs = 5

def train_CIFAR(model, train_loaded, val_loaded, epochs, optimizer, criteria):
  train_l = []
  val_l = []
  train_acc = []
  val_acc = []

  for epoch in range(epochs):
    model.train()
    run_l = 0
    c = 0
    n = 0

    for image, label in tqdm.tqdm(train_loaded, unit="batch"):
      image, label = image.to(device), label.to(device)
      optimizer.zero_grad()
      output = model(image)
      loss = criteria(output, label)
      loss.backward()
      optimizer.step()

      run_l += loss.item()
      i, pred = torch.max(output, 1)
      c += (pred == label).sum().item()
```

```
        n += label.size(0)

    train_l.append(run_l / len(train_loaded))
    train_acc.append(c / n)

    model.eval()
    val_loss = 0
    val_c = 0
    val_n = 0

    with torch.no_grad():
      for image, label in val_loaded:
        image, label = image.to(device), label.to(device)
        output = model(image)
        loss = criteria(output, label)

        val_loss += loss.item()
        i, pred = torch.max(output, 1)
        val_c += (pred == label).sum().item()
        val_n += label.size(0)

    val_l.append(val_loss / len(val_loaded))
    val_acc.append(val_c / val_n)


  return train_l, val_l, train_acc, val_acc
```

```
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is depr
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/alexnet-owt-7be5be79.pth" to /root/.cache/torch/hub/checkpoints/alexnet-
100%|██████████| 233M/233M [00:01<00:00, 182MB/s]
```

```
train_ls, val_ls, train_accs, val_accs = train_CIFAR(model, train_loaded, val_loaded, epochs, optimizer, criteria)
```

```
100%|██████████| 625/625 [01:35<00:00,  6.54batch/s]
100%|██████████| 625/625 [01:37<00:00,  6.44batch/s]
100%|██████████| 625/625 [01:41<00:00,  6.19batch/s]
100%|██████████| 625/625 [01:35<00:00,  6.51batch/s]
100%|██████████| 625/625 [01:34<00:00,  6.62batch/s]
```

```
print(num_trainable_params(model))
```

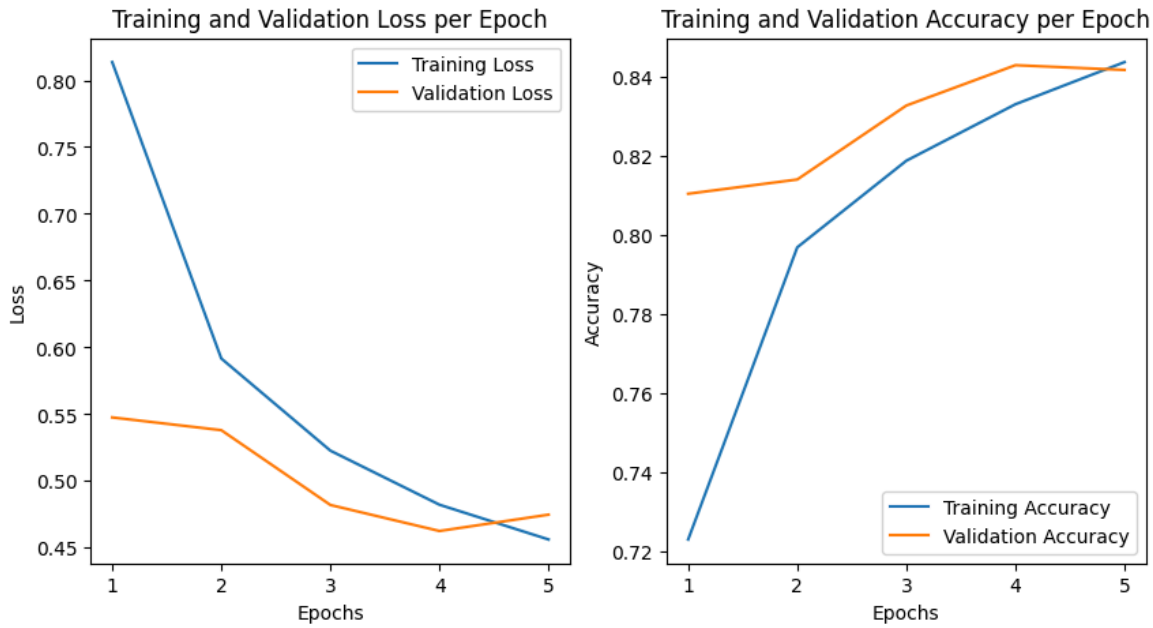
```
41958410
```

```
ep_range = range(1, 6)

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.plot(ep_range, train_ls, label='Training Loss')
plt.plot(ep_range, val_ls, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Training and Validation Loss per Epoch')

plt.subplot(1, 2, 2)
plt.plot(ep_range, train_accs, label='Training Accuracy')
plt.plot(ep_range, val_accs, label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training and Validation Accuracy per Epoch')

plt.show()
```



Kaggle Submission

The following code is for you to make your submission to kaggle. Here are the steps you must follow:

1. Upload `cifar-test-data-sp25.npy` to the colab notebook by going to files on the left hand pane, then hitting "upload". This file may take roughly a minute to upload and you should not proceed to the following steps until it has completely finished uploading (students in the past have run into issues where they were accidentally testing on a partially uploaded test set and getting garbage results).
2. Run the following cell to generate the dataset object for the test data. Feel free to modify the code to use the same transforms that you use for the training data. By default, this will re-use the `transform` variable.
3. In the second cell, write code to run predictions on the testing dataset and store them into an array called `predictions`.
4. Run the final cell which will convert your predictions array into a CSV for kaggle.
5. Go to the files pane again, and download the file called `submission.csv` by clicking the three dots and then download.

```
from PIL import Image
import os

class CIFAR10Test(torchvision.datasets.VisionDataset):

    def __init__(self, transform=None, target_transform=None):
        super(CIFAR10Test, self).__init__(None, transform=transform,
                                          target_transform=target_transform)
        assert os.path.exists("cifar10-test-data-sp25.npy"), "You must upload the test data to the file system."
        self.data = [np.load("cifar10-test-data-sp25.npy", allow_pickle=False)]

        self.data = np.vstack(self.data).reshape(-1, 3, 32, 32)
        self.data = self.data.transpose((0, 2, 3, 1))  # convert to HWC

    def __getitem__(self, index: int):
        img = self.data[index]
        img = Image.fromarray(img)
        if self.transform is not None:
            img = self.transform(img)
        return img

    def __len__(self) -> int:
        return len(self.data)

# To save some hassle, we have provided the same transformation to be applied in the training data, but please
# change this if you were to modify your training approach compared to what we have.
transform = transforms.Compose([
    transforms.Resize((224, 224)), # Resize to 224x224 (height x width)
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
```

```
# Create the test dataset
testing_data = CIFAR10Test(
    transform=transform, # NOTE: Make sure transform is the same as used in the training dataset.
)


### YOUR CODE HERE ###

# Recommendation: create a `test_dataloader` from torch.utils.data.DataLoader with `shuffle=False` to iterate over the test d
test_loaded = DataLoader(testing_data, batch_size=64, shuffle=False)

# Store a numpy vector of the predictions for the test set in the variable `predictions`.
model.eval()
predictions = []

with torch.no_grad():
    for batch in tqdm.tqdm(test_loaded, unit="batch"):
        inputs = batch if not isinstance(batch, (list, tuple)) else batch[0]
        inputs = inputs.to(device)

        if inputs.dim() == 3:
            inputs = inputs.unsqueeze(0)

        outputs = model(inputs)
        preds = outputs.argmax(dim=1)
        predictions.extend(preds.cpu().numpy())

predictions = np.array(predictions)
```

```
100%|██████████| 157/157 [00:21<00:00,  7.43batch/s]
```

```
# This code below will generate kaggle_predictions.csv file. Please download it and submit to kaggle.
import pandas as pd

if isinstance(predictions, np.ndarray):
    predictions = predictions.astype(int)
else:
    predictions = np.array(predictions, dtype=int)
assert predictions.shape == (len(testing_data),), "Predictions were not the correct shape"
df = pd.DataFrame({'Category': predictions})
df.index += 1  # Ensures that the index starts at 1.
df.to_csv('submission.csv', index_label='Id')

# Now download the submission.csv file to submit.


#7.4 unfrozen model
class TransferCIFAR10Unfrozen(nn.Module):
    def __init__(self):
        super(TransferCIFAR10Unfrozen, self).__init__()
        self.model = models.alexnet(pretrained=True)

        self.model.classifier = nn.Sequential(
            nn.Dropout(0.5),
            nn.Linear(256 * 6 * 6, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(0.5),
            nn.Linear(4096, 1024),
            nn.ReLU(inplace=True),
            nn.Linear(1024, 10)
        )

    def forward(self, x):
        return self.model(x)

unfrozen_model = TransferCIFAR10Unfrozen().to(device)
```

```
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is depr
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum
  warnings.warn(msg)
```

```
criteria = nn.CrossEntropyLoss()
optimizer_unfrozen = torch.optim.Adam(unfrozen_model.parameters(), lr=0.001)
```

```
100%|██████████| 625/625 [01:40<00:00,  6.24batch/s]
100%|██████████| 625/625 [01:40<00:00,  6.22batch/s]
100%|██████████| 625/625 [01:41<00:00,  6.14batch/s]
100%|██████████| 625/625 [01:40<00:00,  6.20batch/s]
100%|██████████| 625/625 [01:40<00:00,  6.20batch/s]
```

```
print(num_trainable_params(unfrozen_model))
```

```
44428106
```

```
ep_range = range(1, 6)

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.plot(ep_range, train_ls_unfrozen, label='Training Loss')
plt.plot(ep_range, val_ls_unfrozen, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Unfrozen Training and Validation Loss per Epoch')

plt.subplot(1, 2, 2)
plt.plot(ep_range, train_accs_unfrozen, label='Training Accuracy')
plt.plot(ep_range, val_accs_unfrozen, label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
```