KOREATECH
한국기술교육대학교

Transfer Learning

December 2023

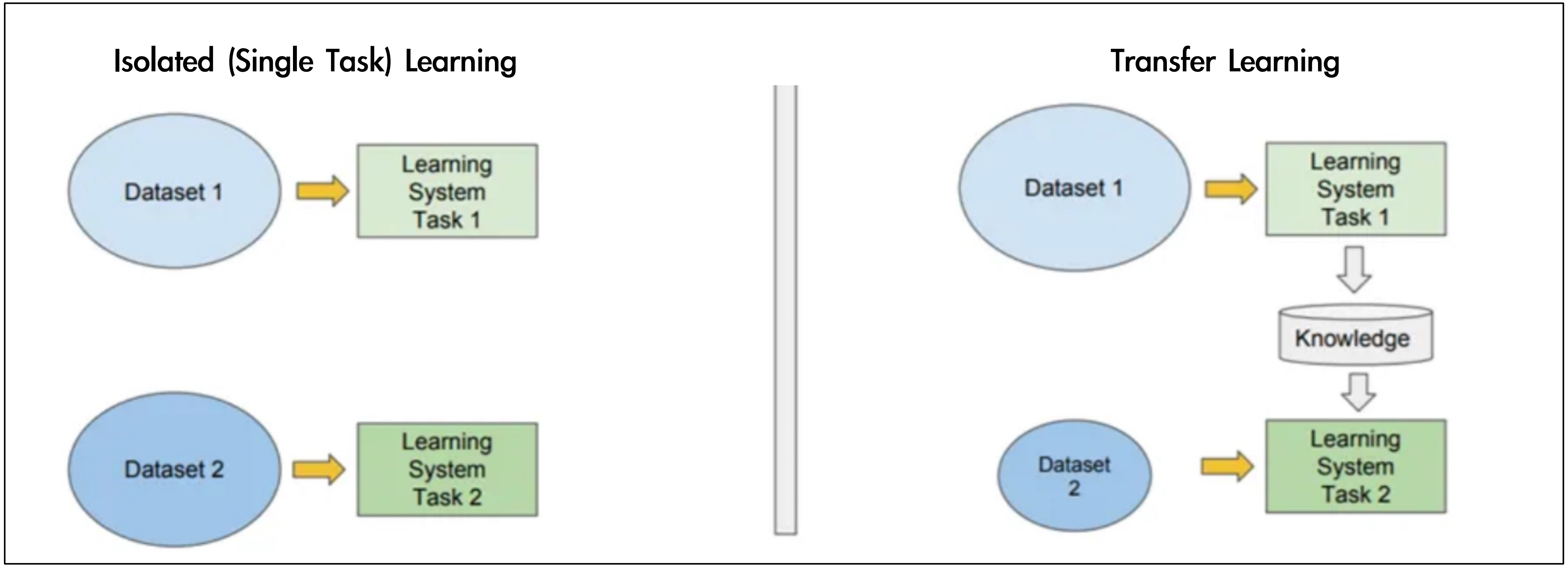
http://link.koreatech.ac.kr

Transfer Learning

Transfer Learning

◈What is Transfer Learning?

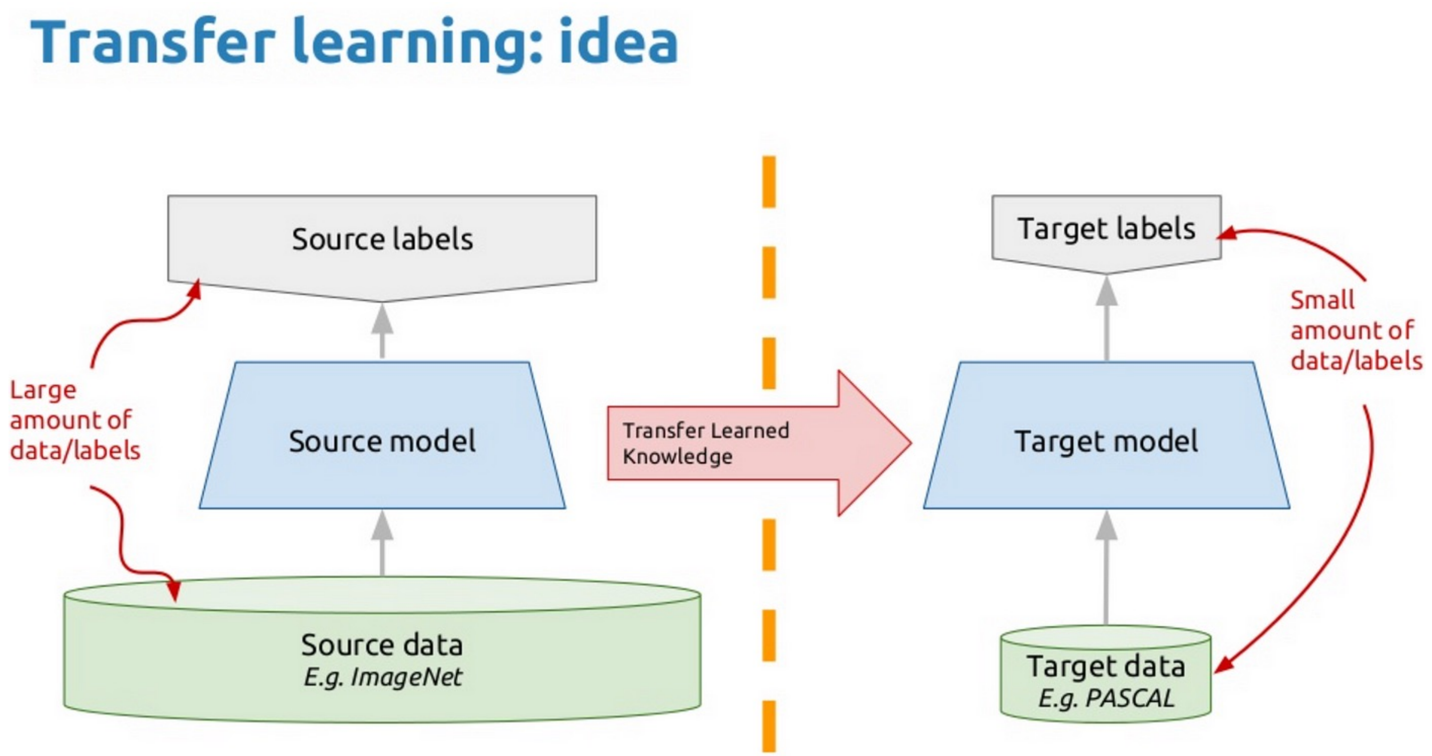
- Transfering the knowledge of one model to perform a new task
- Reuse of a pre-trained model on a new task
- a.k.a "Domain Adaptation"



Transfer Learning

◈Motivation of Transfer Learning

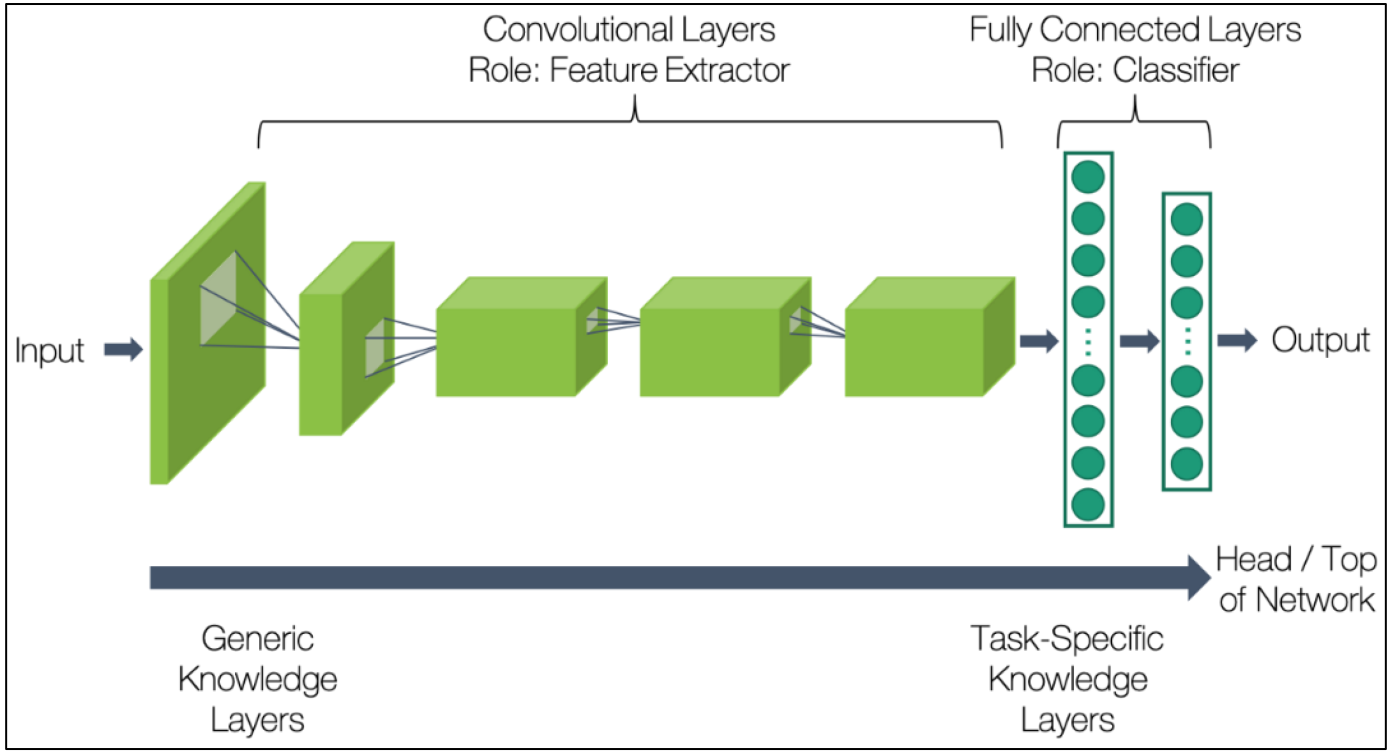
- Lots of data, time, resources needed to train and tune a neural network from scratch
  - An ImageNet deep neural network can take weeks to train and fine-tune from scratch
  - Unless you have 256 GPUs, it is not possible to achieve in 1 hour
- Cheaper & faster way of adapting a neural network by <u>exploiting their generalization properties</u>

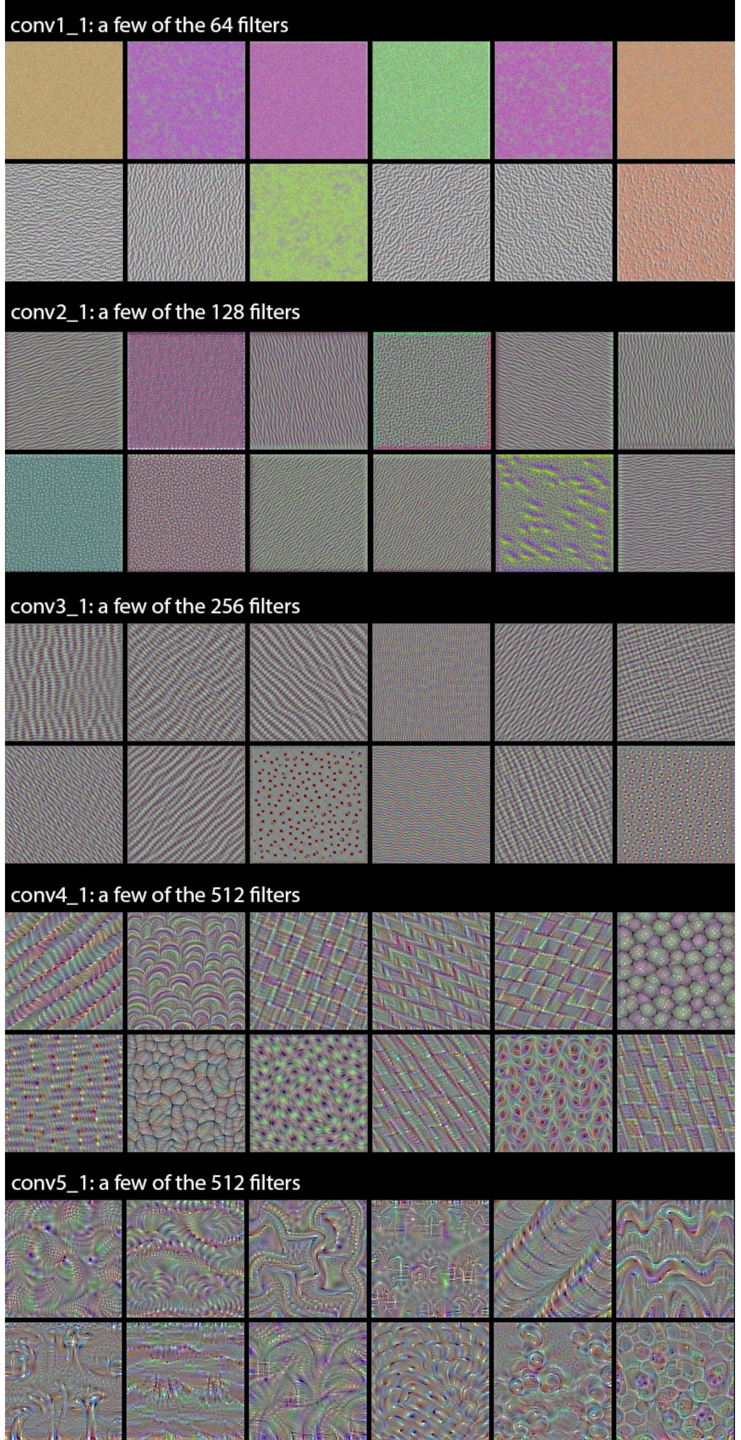


Transfer Learning

◈Neural Network Layers: General to Specific

- Bottom/first/earlier layers: general learners
  - Low-level notions of edges, visual shapes
- Top/last/later layers: specific learners
  - High-level features such as eyes, feathers





Example: VGG 16 Filters

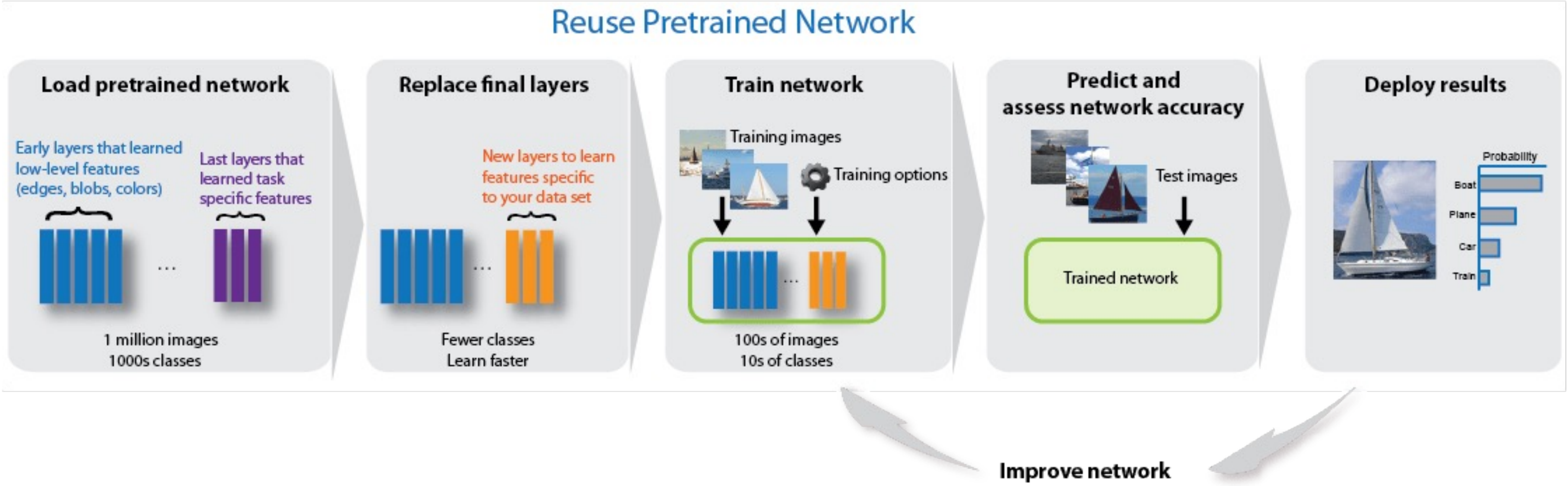
Transfer Learning

◈Transfer Learning Process (1/2)

– 1. Start with a pre-trained network

– 2. Partition network into:

- Featurizers: identify which layers to keep (feature extraction layers)
- Classifiers: identify which layers to replace
  - ➢ In many cases, we replace the old layers with new ones (with different numbers of outputs) on top of featurizers



Transfer Learning

◈Transfer Learning Process (2/2)

– 3. Two major transfer learning methods

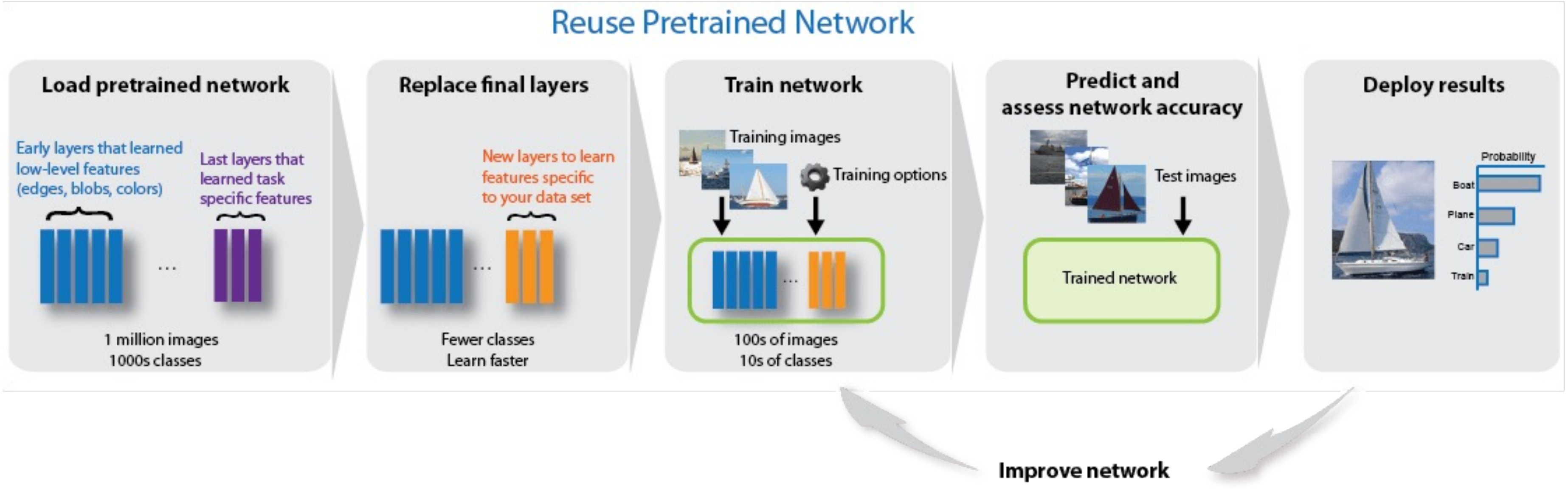
1) Frozen the featurizers and Train the classifiers (including new layers) with new data

➢ It is common to pretrain a ConvNet on a very large dataset (e.g. ImageNet, which contains 1.2 million images with 1000 categories), and then use the ConvNet as a featurizers for the task of interest

2) Fine-tune the whole model with new data

➢ Unfreeze weights and fine-tune the whole network with smaller learning rate

➢ It is not common method because it is relatively rare to have a dataset of sufficient size

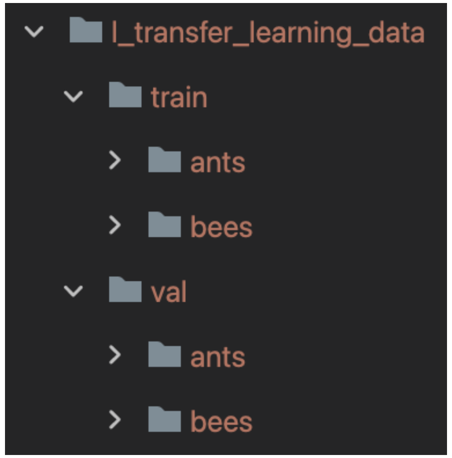


Transfer Learning with PyTorch

Data Preparation

◈Data Preparation

– We will use torchvision and torch.utils.data packages for loading the data
– The problem we're going to solve today is to train a model to classify ants and bees
  • There are about 120 training images each for ants and bees
  • There are 75 validation images each for ants and bees
– Usually, this is a very small dataset to generalize upon, if trained from scratch.
– Since we are using transfer learning, we should be able to generalize reasonably well
– Download the data from here and extract it to the predefined folder
  • https://download.pytorch.org/tutorial/hymenoptera_data.zip
  • Make "_00_data/l_transfer_learning_data" and store the unzipped files into the new folder



Transfer Learning with PyTorch

◈Preparation

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
import numpy as np
from torchvision import datasets, models, transforms
import matplotlib.pyplot as plt
import time, os, sys
from pathlib import Path

BASE_PATH = str(Path(__file__).resolve().parent.parent.parent)  # BASE_PATH: /Users/yhhan/git/link_dl
sys.path.append(BASE_PATH)

CURRENT_FILE_PATH = os.path.dirname(os.path.abspath(__file__))
CHECKPOINT_FILE_PATH = os.path.join(CURRENT_FILE_PATH, "checkpoints")

if not os.path.isdir(CHECKPOINT_FILE_PATH):
  os.makedirs(os.path.join(CURRENT_FILE_PATH, "checkpoints"))
```

Transfer Learning with PyTorch

◈Data Transforms

```
data_transforms = {
  'train': transforms.Compose([
    transforms.RandomResizedCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
  ]),

  'val': transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
  ])
}
```

link_dl/_01_code/_12_transfer_learning/a_train_transfer_learning.py

Transfer Learning with PyTorch

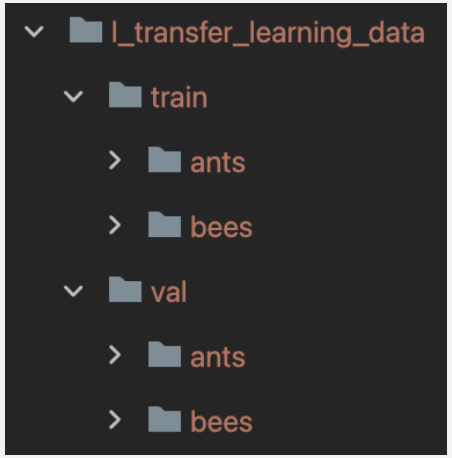
◈ Get New Data

```
def get_new_data():
  new_data_path = os.path.join(BASE_PATH, "_00_data", "l_transfer_learning_data")
  image_datasets = {
    x: datasets.ImageFolder(os.path.join(new_data_path, x), data_transforms[x])
    for x in ['train', 'val']
  }

  dataset_sizes = {
    x: len(image_datasets[x]) for x in ['train', 'val']
  }

  dataloaders = {
    x: torch.utils.data.DataLoader(image_datasets[x], batch_size=4, shuffle=True, num_workers=0)
    for x in ['train', 'val']
  }

  class_names = image_datasets['train'].classes
  return image_datasets, dataset_sizes, dataloaders, class_names
```



Transfer Learning with PyTorch

◈Train

```
def train_model(
  dataloaders, dataset_sizes, model, loss_fn, optimizer, scheduler, num_epochs=25, device=torch.device("cpu")
):
  since = time.time()

  best_model_params_path = os.path.join(CHECKPOINT_FILE_PATH, 'best_model_params.pt')

  best_acc = 0.0

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

    # Each epoch has a training and validation phase
    for phase in ['train', 'val']:
      if phase == 'train':
        model.train()  # Set model to training mode
      else:
        model.eval()  # Set model to evaluate mode
```

Transfer Learning with PyTorch

◈Train

```
def train_model(
  dataloaders, dataset_sizes, model, loss_fn, optimizer, scheduler, num_epochs=25, device=torch.device("cpu")
):
  ...
  for epoch in range(num_epochs):
    ...
    for phase in ['train', 'val']:
      ...
      running_loss = 0.0
      running_corrects = 0

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

        # zero the parameter gradients
        optimizer.zero_grad()
```

Transfer Learning with PyTorch

◈Train

```
def train_model(
  dataloaders, dataset_sizes, model, loss_fn, optimizer, scheduler, num_epochs=25, device=torch.device("cpu")
):
  ...
  for epoch in range(num_epochs):
    ...
    for phase in ['train', 'val']:
      ...
      for inputs, labels in dataloaders[phase]:
        ...
        # track history if only in train
        with torch.set_grad_enabled(phase == 'train'):
          outputs = model(inputs)
          _, preds = torch.max(outputs, 1)
          loss = criterion(outputs, labels)
          # backward + optimize only if in training phase
          if phase == 'train':
            loss.backward()
            optimizer.step()
```

Transfer Learning with PyTorch

◈Train

```
def train_model(
  dataloaders, dataset_sizes, model, loss_fn, optimizer, scheduler, num_epochs=25, device=torch.device("cpu")
):
  ...
  for epoch in range(num_epochs):
    ...
    for phase in ['train', 'val']:
      ...
      for inputs, labels in dataloaders[phase]:
        ...
        # statistics
        running_loss += loss.item() * inputs.size(0)
        running_corrects += torch.sum(preds == labels.data)

      if phase == 'train':
        scheduler.step()  # Scheduling the learning rate

      epoch_loss = running_loss / dataset_sizes[phase]
      epoch_acc = running_corrects.double() / dataset_sizes[phase]
```

Transfer Learning with PyTorch

◈Train

```
def train_model(
  dataloaders, dataset_sizes, model, loss_fn, optimizer, scheduler, num_epochs=25, device=torch.device("cpu")
):
  ...
  for epoch in range(num_epochs):
    ...
    for phase in ['train', 'val']:
      ...
      print(f'{phase} Loss: {epoch_loss:.4f} Acc: {epoch_acc:.4f}')
      # deep copy the model
      if phase == 'val' and epoch_acc > best_acc:
        best_acc = epoch_acc
        torch.save(model.state_dict(), best_model_params_path)

    print()
  time_elapsed = time.time() - since
  print(f'Training complete in {time_elapsed // 60:.0f}m {time_elapsed % 60:.0f}s')
  print(f'Best val Acc: {best_acc:4f}\n')
```

Transfer Learning with PyTorch

◈Train

```
def train_model(
  dataloaders, dataset_sizes, model, loss_fn, optimizer, scheduler, num_epochs=25, device=torch.device("cpu")
):
  ...

  # load best model weights
  model.load_state_dict(torch.load(best_model_params_path))

  return model
```

Transfer Learning with PyTorch

◈Visualization

```
def imshow(input, label=None):
  """Display image for Tensor."""
  input = input.numpy().transpose((1, 2, 0))

  mean = np.array([0.485, 0.456, 0.406])
  std = np.array([0.229, 0.224, 0.225])

  input = std * input + mean
  input = np.clip(input, 0, 1)

  plt.imshow(input)

  if label is not None:
    plt.title(label)

  plt.show()
```

Transfer Learning with PyTorch

◈Visualization

```
def visualize_model(dataloaders, class_names, model, num_images=6):
  model.eval()

  images_so_far = 0

  for i, (inputs, labels) in enumerate(dataloaders['val']):
    outputs = model(inputs)
    _, preds = torch.max(outputs, dim=1)

    for j in range(inputs.size()[0]):
      imshow(
        inputs.data[j], label="Prediction: {0} - Label: {1}".format(
          class_names[preds[j].item()], class_names[labels[j].item()]
        )
      )
      images_so_far += 1
      if images_so_far == num_images:
        return
```

Transfer Learning with PyTorch

◈Get Model

```
def get_model(method, device=torch.device("cpu")):
  model_ft = models.resnet18(weights='IMAGENET1K_V1')

  if method == "frozen_and_train_new_classifier":
    for param in model_ft.parameters():
      param.requires_grad = False

  print(model_ft)
  print("#" * 100)

  # Here the size of each output sample is set to 2
  num_ftrs = model_ft.fc.in_features
  model_ft.fc = nn.Linear(in_features=num_ftrs, out_features=2)
  print(model_ft)
  print("#" * 100)

  model_ft = model_ft.to(device)
  return model_ft
```

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

Transfer Learning with PyTorch

◈Main

```
def main(method):
  device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

  image_datasets, dataset_sizes, dataloaders, class_names = get_new_data()
  model_ft = get_model(method, device)
  loss_fn = nn.CrossEntropyLoss()

  # Observe that all parameters are being optimized
  optimizer_ft = optim.Adam(model_ft.parameters(), lr=0.001)

  # Decay LR by a factor of 0.1 every 7 epochs
  exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=7, gamma=0.1)

  model_ft = train_model(
    dataloaders, dataset_sizes, model_ft, loss_fn, optimizer_ft, exp_lr_scheduler,
    num_epochs=25, device=device
  )
  visualize_model(dataloaders, class_names, model_ft)
```

Transfer Learning with PyTorch

◈Main

```
if __name__ == "__main__":
    method_idx = 0
    methods = [
        "frozen_and_train_new_classifier",
        "fine_tune_the_whole_model"
    ]
    main(methods[method_idx])
```

```
...
...

Epoch 24/24
----------
train Loss: 0.4411 Acc: 0.7951
val Loss: 0.2509 Acc: 0.9346

Training complete in 2m 28s
Best val Acc: 0.941176
```

```
if __name__ == "__main__":
    method_idx = 1
    methods = [
        "frozen_and_train_new_classifier",
        "fine_tune_the_whole_model"
    ]
    main(methods[method_idx])
```

```
...
...

Epoch 24/24
----------
train Loss: 0.6055 Acc: 0.6721
val Loss: 0.5691 Acc: 0.6732

Training complete in 5m 26s
Best val Acc: 0.699346
```