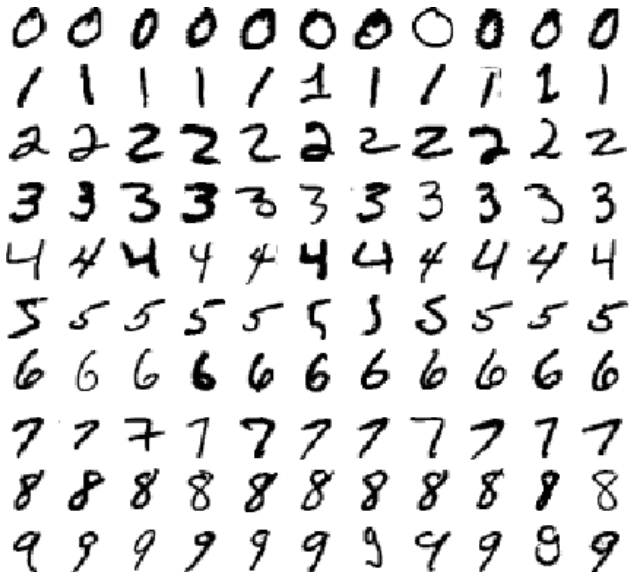
Practice with CNNs

Machine Intelligence Lab
Handong Global University

Practice – MNIST



MNIST dataset [28X28] = [1 X 784]

Hand Written Digit Number from 0 to 9
Data includes data and label.

Pytorch Dataset(torchvision) provides
50,000 images to train,
10,000 images to test.

model

```
[1]  from google.colab import drive
     drive.mount('/content/drive')
```

```
[4]  from __future__ import print_function
     import argparse
     import torch
     import torch.nn as nn
     import torch.nn.functional as F
     import torch.optim as optim
     from torchvision import datasets, transforms
```

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(784, 500)
        self.fc2 = nn.Linear(500, 300)
        self.fc3 = nn.Linear(300, 100)
        self.fc4 = nn.Linear(100, 10)

    def forward(self, x):
        batch_size, c, h, w = x.data.size()
        x = x.view(batch_size, 784)
        x = torch.tanh(self.fc1(x))
        x = torch.tanh(self.fc2(x))
        x = torch.tanh(self.fc3(x))
        x = self.fc4(x)
        return F.log_softmax(x, dim=1)
```

Conv2D(in_channels, out_channels, kernel_size, ...)

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(1, 20, 5, 1),
            nn.ReLU(),
            nn.MaxPool2d(2,2),
            nn.Conv2d(20, 50, 5, 1),
            nn.ReLU(),
            nn.MaxPool2d(2,2)
        )
        conv_size = self.get_conv_size((1, 28, 28))
        self.fc = nn.Sequential(
            nn.Linear(conv_size, 500), # conv_size = 4*4*50
            nn.Linear(500, 10)
        )

    def get_conv_size(self, shape):
        o = self.conv(torch.zeros(1, *shape))
        return int(np.prod(o.size()))

    def forward(self, x):
        batch_size, c, h, w = x.data.size() # 32*1*28*28
        x = self.conv(x)
        x = x.view(batch_size, -1) # conv_size = 4*4*50
        x = self.fc(x)
        return F.log_softmax(x, dim=1)
```

functions: train and test

```python
def train(model, device, train_loader, optimizer, epoch, log_interval):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()

        if batch_idx % log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))

            #torch.save(model.state_dict(),"drive/My Drive/public/results/mnist_cnn.pt")
```

```python
def test(model, device, test_loader):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)

            # sum up batch loss
            test_loss += F.nll_loss(output, target, reduction='sum').item()

            # get the index of the max log-probability
            pred = output.argmax(dim=1, keepdim=True)
            correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss /= len(test_loader.dataset)

    print('\nTest: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)))
```

config and data

```
seed = 1
epochs = 2
batch_size = 32
test_batch_size = 1000
lr = 0.001
momentum = 0.9
log_interval = 100
save_model = True

torch.manual_seed(seed)

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

kwargs = {'num_workers': 1, 'pin_memory': True} if torch.cuda.is_available() else {}
transform=transforms.Compose([
   transforms.ToTensor(),
   transforms.Normalize((0.1307,), (0.3081,)) ])
train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('drive/My Drive/public/data', train=True,
                   download=True, transform=transform),
    batch_size=batch_size, shuffle=True, **kwargs)
test_loader = torch.utils.data.DataLoader(
    datasets.MNIST('drive/My Drive/public/data', train=False,
                   transform=transform),
    batch_size=test_batch_size, shuffle=True, **kwargs)
```

optimize and train

```
model = Net().to(device)
optimizer = optim.SGD(model.parameters(), lr=lr, momentum=momentum)

for epoch in range(1, epochs + 1):
    train(model, device, train_loader, optimizer, epoch, log_interval)
    test(model, device, test_loader)

if (save_model):
    if not os.path.exists('drive/My Drive/public/results'):
        os.mkdir('drive/My Drive/public/results')
    torch.save(model,"drive/My Drive/public/results/mnist_cnn.pth")
```

check: save/load model

```
torch.save(model,"drive/My Drive/public/results/mnist_cnn.pth")
```

```
!ls "drive/My Drive/public/results/"
```

```
cifar10_model.pth        imagenet_nasnetalarge.pth   ptb_trained_model.pth
cifar10_pretrained.pth   mnist_cnn.pth               train_log.txt
cifar_model.pkl          ptb_trained_model_best.pth
```

```
load_model = torch.load("drive/My Drive/public/results/mnist_cnn.pth")
```

CIFAR10 Example

10 classes

airplane

automobile

bird

cat

deer

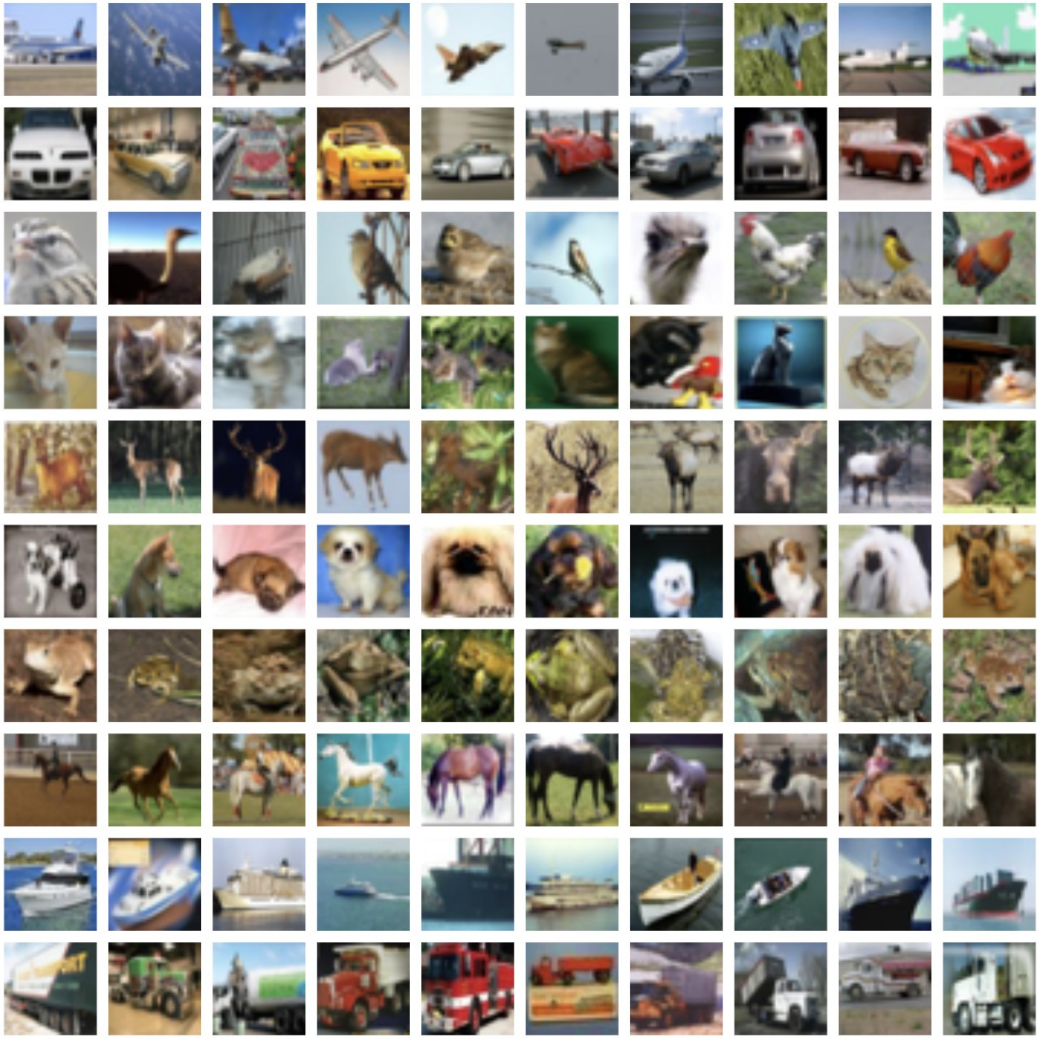
dog

frog

horse

ship

truck



CIFAR10 Example

- CIFAR10

mount your drive

```
[5]  from google.colab import drive
     drive.mount('/content/drive')
```

import pytorch

```
[6]  import numpy as np
     import os
     import torch
     import torch.nn as nn
     import torch.optim as optim
     import torch.nn.init as init
     import torchvision.datasets as dset
     import torchvision.transforms as transforms
     from torch.utils.data import DataLoader
     from torch.autograd import Variable

     import warnings
     warnings.filterwarnings('ignore')
```

CIFAR10 Example: model architecture

- CIFAR10

  32 x 32 x 3

MNIST => 28 x 28 x 1

CIFAR10 => 32 x 32 x 3

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN,self).__init__()
        self.layer = nn.Sequential(
            nn.Conv2d(3,16,3,padding=1),
            nn.ReLU(),
            nn.Conv2d(16,32,3,padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2,2), # 32 x 16 x 16

            nn.Conv2d(32,64,3,padding=1),
            nn.ReLU(),
            nn.Conv2d(64,128,3,padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2,2), # 128 x 8 x 8

            nn.Conv2d(128,256,3,padding=1),
            nn.ReLU(),
            nn.Conv2d(256,256,3,padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2,2)
        )

        conv_size = self.get_conv_size((3,32,32))

        self.fc_layer = nn.Sequential(
            nn.Linear(conv_size,200),
            nn.ReLU(),
            nn.Linear(200,10)
        )
```

```
def get_conv_size(self, shape):
    o = self.layer(torch.zeros(1, *shape))
    return int(np.prod(o.size()))

def forward(self,x):
    batch_size, c, h, w = x.data.size()
    out = self.layer(x)
    out = out.view(batch_size, -1)
    out = self.fc_layer(out)
    return out
```

CIFAR10 Example: data and loader

- CIFAR10

```
batch_size = 32

cifar_train = dset.CIFAR10("drive/My Drive/public/data/", train=True,
                           transform=transforms.ToTensor(),
                           target_transform=None, download=True)
cifar_test = dset.CIFAR10('drive/My Drive/public/data/", train=False,
                          transform=transforms.ToTensor(),
                          target_transform=None, download=True)

print(cifar_train.__getitem__(0)[0].size(), cifar_train.__len__())
cifar_test.__getitem__(0)[0].size(), cifar_test.__len__()

train_loader = torch.utils.data.DataLoader(cifar_train,batch_size=batch_size,
                                           shuffle=True,num_workers=2,drop_last=True)
test_loader = torch.utils.data.DataLoader(cifar_test,batch_size=batch_size,
                                          shuffle=False,num_workers=2,drop_last=True)
```

```
Files already downloaded and verified
Files already downloaded and verified
torch.Size([3, 32, 32]) 50000
```

CIFAR10 Example: training

```
learning_rate = 0.0002
num_epoch = 10

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

model = CNN().to(device)
loss_func = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

```
try:
    model = torch.load('drive/My Drive/public/results/cifar10_pretrained.pth')
    print("model restored")
except:
    print("model not restored")

if not os.path.exists('drive/My Drive/public/results'):
    os.mkdir('drive/My Drive/public/results')

for i in range(num_epoch):
    for j,[image,label] in enumerate(train_loader):
        x = Variable(image).to(device)
        y_= Variable(label).to(device)

        optimizer.zero_grad()
        output = model.forward(x)
        loss = loss_func(output,y_)
        loss.backward()
        optimizer.step()

        if j % 500 == 0:
            print(loss.data,i,j)

    torch.save(model,'drive/My Drive/public/results/cifar10_model.pth')
```

```
param_list = list(model.children())
print(param_list)
```

```
tensor(0.9526, device='cuda:0') 3 500
tensor(0.7414, device='cuda:0') 3 1000
tensor(0.8229, device='cuda:0') 3 1500
tensor(0.7450, device='cuda:0') 4 0
tensor(0.8136, device='cuda:0') 4 500
tensor(0.7042, device='cuda:0') 4 1000
tensor(0.8028, device='cuda:0') 4 1500
tensor(0.8475, device='cuda:0') 5 0
tensor(0.6328, device='cuda:0') 5 500
tensor(0.9459, device='cuda:0') 5 1000
tensor(0.6946, device='cuda:0') 5 1500
tensor(0.4617, device='cuda:0') 6 0
tensor(0.6533, device='cuda:0') 6 500
tensor(0.6549, device='cuda:0') 6 1000
tensor(0.9481, device='cuda:0') 6 1500
tensor(0.3352, device='cuda:0') 7 0
tensor(0.8327, device='cuda:0') 7 500
tensor(0.6962, device='cuda:0') 7 1000
tensor(0.4082, device='cuda:0') 7 1500
```

CIFAR10 Example: testing

```
correct = 0
total = 0

with torch.no_grad():
  for image,label in test_loader:
    x = Variable(image).to(device)
    y_= Variable(label).to(device)

    output = model.forward(x)
    _,output_index = torch.max(output,1)

    total += label.size(0)
    correct += (output_index == y_).sum().float()

print("Accuracy of Test Data: {}".format(100*correct/total))

torch.save(model, 'drive/My Drive/public/results/cifar10_pretrained.pth')
```

```
Accuracy of Test Data: 73.34735870361328
```

Practice: CIFAR10 Example

- Try to change the network architecture
  - the number of layers, the number of kernels
  - padding, stride and so on

- Improve your model's performance

Training your own data example

- put images in the class directories for train, valid and test.
- for example, with 'HDH' and 'OH' classes
  - ./drive/My Drive/public/train/HDH/*.jpg
  - ./drive/My Drive/public/train/OH/*.jpg
  - ./drive/My Drive/public/valid/HDH/*.jpg
  - ./drive/My Drive/public/valid/OH/*.jpg

model architecture

- We need a more complicated model

```
class MyCNN(nn.Module):
    def __init__(self, output_dim=10):
        super(MyCNN,self).__init__()

        self.output_dim=output_dim

        self.cnn_layers = nn.Sequential(
            nn.Conv2d(3,32,3,padding=1), # try with different kernels
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.Conv2d(32,32,3,padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(4,4), # 32 x (25x25)

            nn.Conv2d(32,16,3,padding=1),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.Conv2d(16,16,3,padding=1),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.MaxPool2d(5,5) # 16 x (5x5)
        )
        conv_size = self.get_conv_size((3, 100, 100))
        self.fc_layer = nn.Sequential(
            nn.Linear(16*5*5,100),
            nn.BatchNorm1d(100),
            nn.ReLU(),
            nn.Linear(100,output_dim)
        )
```

```
def get_conv_size(self, shape):
    o = self.cnn_layers(torch.zeros(1, *shape))
    return int(np.prod(o.size()))

def forward(self,x):
    batch_size, c, h, w = x.data.size()
    out = self.cnn_layers(x)
    out = out.view(batch_size, -1)
    out = self.fc_layer(out)
    return out
```

Transform function

```
learning_rate = 0.0005
output_dim=5
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model = MyCNN(output_dim=output_dim).to(device)
loss_func = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

param_list = list(model.children())
print(param_list)
```

```
resize=(120, 120)
input_size=(100, 100)

data_transforms = {
    'train': transforms.Compose([
        transforms.Resize(resize),
        transforms.RandomCrop(input_size), # data augmentation
        transforms.ToTensor(),
        transforms.Normalize([0.5, 0.5, 0.5], [0.3, 0.3, 0.3])
    ]),
    'valid': transforms.Compose([
        transforms.Resize(resize),
        transforms.CenterCrop(input_size),
        transforms.ToTensor(),
        transforms.Normalize([0.5, 0.5, 0.5], [0.3, 0.3, 0.3])
    ]),
}
test_transform = data_transforms['valid']
```

Data Loader

```
batch_size = 64 # try different batch_size values
data_dir = './drive/My Drive/public/data/'
train_dir = 'train'
valid_dir = 'valid'

train_set = datasets.ImageFolder(data_dir+train_dir, data_transforms['train'])
valid_set = datasets.ImageFolder(data_dir+valid_dir, data_transforms['valid'])

train_loader = torch.utils.data.DataLoader(train_set, batch_size=batch_size,
                                           shuffle=True, num_workers=4)
valid_loader = torch.utils.data.DataLoader(valid_set, batch_size=batch_size,
                                           shuffle=True, num_workers=4)

train_size = len(train_set)
valid_size = len(valid_set)

class_names = train_set.classes

print(class_names)
print(f'Train image size: {train_size}')
print(f'Validation image size: {valid_size}')
```

Training

- the same as the previous slides

```
result_dir = 'drive/My Drive/public/results/'
num_epoch = 3 # try with different epochs and find the best epoch

if not os.path.exists(result_dir):
    os.mkdir(result_dir)

for i in range(num_epoch):
    model.train()
    for j, [image,label] in enumerate(train_loader):
        x = image.to(device)
        y_= label.to(device)

        optimizer.zero_grad()
        output = model(x)
        loss = loss_func(output,y_)
        loss.backward()
        optimizer.step()

        if j % 30 == 0:
            print(i,j, loss.data.cpu())

    model.eval()
    hits = 0
    for k,[image,label] in enumerate(valid_loader):
        x = image.to(device)
        y_= label.to(device)

        output = model(x)
        y_est = output.argmax(1)
        hits = hits + sum(y_est == y_).cpu()
    print('Hits', int(hits), 'Accuracy', float(hits/(valid_size+0.0)))

torch.save(model, result_dir + 'teamX.model')
print('training is done by max_epochs', num_epoch)
```

Testing

- the same as the previous slides

```
test_batch_size = 10
result_dir = 'drive/My Drive/public/results/'
model_name = 'teamX.model'

model = torch.load(result_dir + model_name)
model.to(device)
model.eval()

test_dir = './drive/My Drive/public/data/valid'
test_set = datasets.ImageFolder(test_dir, test_transform)

test_loader = torch.utils.data.DataLoader(test_set, batch_size=test_batch_size,
                                          shuffle=False, num_workers=4)

hits = 0
for k,[image,label] in enumerate(test_loader):
    x = image.to(device)
    y_= label.to(device)

    output = model(x)
    y_est = output.argmax(1)
    print('Target', label.numpy(), 'Prediction', y_est.cpu().numpy())
    hits = hits + sum(y_est == y_)
print('hits', int(hits),'accuracy', float(hits/(len(test_set)+0.0)))
```

```
Target [3 3 3 3 3 3 3 3 3 3] Prediction [3 3 3 3 3 3 3 3 4 3]
Target [3 3 3 3 3 3 3 3 3 3] Prediction [3 3 3 3 3 3 3 3 3 3]
Target [3 3 3 3 3 3 3 3 3 3] Prediction [3 3 3 3 3 3 4 3 4 3]
Target [3 3 3 3 3 3 3 3 3 3] Prediction [3 3 4 3 3 0 0 4 4 1]
Target [4 4 4 4 4 4 4 4 4 4] Prediction [4 4 4 4 4 4 4 4 4 4]
Target [4 4 4 4 4 4 4 4 4 4] Prediction [4 4 4 4 4 4 4 4 4 4]
Target [4 4 4 4 4 4 4 4 4 4] Prediction [4 4 4 4 4 4 4 4 4 4]
Target [4 4 4 4 4 4 4 4 4 4] Prediction [4 4 4 4 4 4 4 4 4 4]
hits 173 accuracy 0.8650000095367432
```

Testing

- the same as the previous slides

```
from skimage import io

img_name = './drive/My Drive/public/data/test/test1.jpg'
test_img = io.imread(img_name)
test_img = transforms.ToPILImage()(test_img)
test_img = test_transform(test_img)
test_data = test_img.reshape(1,3,input_size[0],input_size[1]).to(device)
with torch.no_grad():
  output=model(test_data)

class_id = output.argmax(dim=1).cpu().numpy()[0]
print(img_name.split('/')[-1], '==>', class_id, class_names[class_id])
```

```
from skimage import io
import glob

img_dir = 'drive/My Drive/public/data/test/'
file_list = glob.glob(img_dir + '*.*')
for img_name in file_list:
  test_img = io.imread(img_name)
  test_img = transforms.ToPILImage()(test_img)
  test_img = test_transform(test_img)
  test_data = test_img.reshape(1,3,input_size[0],input_size[1]).to(device)
  with torch.no_grad():
    output=model(test_data)

  class_id = output.argmax(dim=1).cpu().numpy()[0]
  print(img_name.split('/')[-1], '==>', class_id, class_names[class_id])
```

Practice

- Try to change the network architecture
  - the number of layers, the number of kernels
  - padding, stride and so on

- Improve your model's performance

```
class MyCNN(nn.Module):
    def __init__(self, output_dim=10):
        super(MyCNN,self).__init__()

        self.output_dim=output_dim

        self.cnn_layers = nn.Sequential(
            nn.Conv2d(3,32,3,padding=1), # try with different kernels
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.Conv2d(32,32,3,padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(4,4), # 32 x (25x25)

            nn.Conv2d(32,16,3,padding=1),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.Conv2d(16,16,3,padding=1),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.MaxPool2d(5,5) # 16 x (5x5)
        )
        conv_size = self.get_conv_size((3, 100, 100))
        self.fc_layer = nn.Sequential(
            nn.Linear(16*5*5,100),
            nn.BatchNorm1d(100),
            nn.ReLU(),
            nn.Linear(100,output_dim)
```

try to change transform function

- https://pytorch.org/docs/stable/torchvision/transforms.html

```
[ ] input_size=(100, 100)

    data_transforms = {
        'train': transforms.Compose([
            transforms.Resize((120, 120)),
            transforms.RandomCrop(input_size), # data augmentation
            transforms.ToTensor(),
            transforms.Normalize([0.5, 0.5, 0.5], [0.3, 0.3, 0.3])
        ]),
        'valid': transforms.Compose([
            transforms.Resize((120, 120)),
            transforms.CenterCrop(input_size),
            transforms.ToTensor(),
            transforms.Normalize([0.5, 0.5, 0.5], [0.3, 0.3, 0.3])
        ]),
    }
    test_transform = data_transforms['valid']
```

try different learning rates

```
[ ] learning_rate = 0.0005
    output_dim=5
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    model = MyCNN(output_dim=output_dim).to(device)
    loss_func = nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

    param_list = list(model.children())
    print(param_list)
```

ImageNet Challenge

IMAGENET

1.2 Million Images(1,200,000), 1000 Categories

ImageNet Example

- it takes too much time to train.
- so, we will just test with a pretrained model

configuration and loading model

```
from google.colab import drive
drive.mount('/content/drive')
```

```
!pip install pretrainedmodels
!python ./drive/'My Drive'/public/imagenet_install/setup.py install
```

```
import torch
import pretrainedmodels
import pretrainedmodels.utils as utils

model_name = 'nasnetalarge' # could be fbresnet152 or inceptionresnetv2
model = pretrainedmodels.__dict__[model_name](num_classes=1000, pretrained='imagenet')
model.eval()

load_img = utils.LoadImage()

# transformations depending on the model
# rescale, center crop, normalize, and others (ex: ToBGR, ToRange255)
tf_img = utils.TransformImage(model)
```

it might take tens of mins to download the model

Imagenet class names

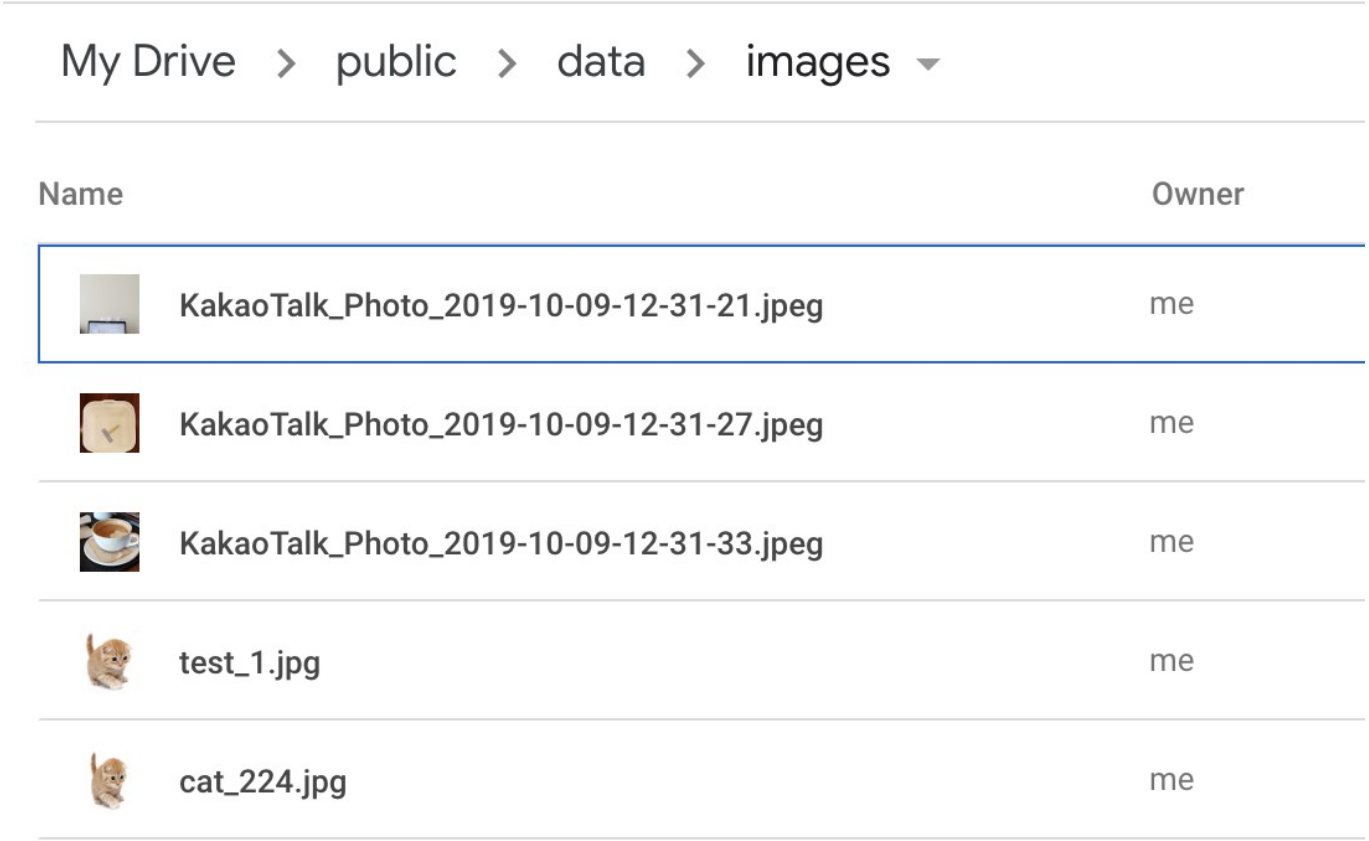
```
import csv
name_file = 'drive/My Drive/public/imagenet_install/class_names.csv'

imagenet_class  = {}
file_in = csv.reader(open(name_file))
for row in file_in:
  imagenet_class[int(row[0])] = row[1]
```

```
print(imagenet_class)
```

```
{0: 'tench, Tinca tinca', 1: 'goldfish, Carassius auratus', 2: 'great wh
```

classification with new images

in my google drive



classification of a single image

```
try:
  model # does exist
except NameError: # model does not exist
  import pretrainedmodels.utils as utils
  model = torch.load('drive/My Drive/public/results/imagenet_nasnetalarge.pth')
  load_img = utils.LoadImage()
  tf_img = utils.TransformImage(model)

# your file name
img_file = './drive/My Drive/public/data/images/cat_224.jpg'

input_img = load_img(img_file)
input_tensor = tf_img(input_img).to(device)            # 3x400x225 -> 3x331x331 size may differ
input_tensor = input_tensor.unsqueeze(0)           # 3x331x331 -> 1x3x331x331

input = torch.autograd.Variable(input_tensor, requires_grad=False)

output_logits = model(input) # 1x1000

print("{} is [{}: {}]".format(img_file ,output_logits.argmax(),
                              imagenet_class[int(output_logits.argmax())]))
```

```
./drive/My Drive/public/data/images/cat_224.jpg is [282: tiger cat]
```

classification of many images in a directory

```
try:
  model # does exist
except NameError: # model does not exist
  import pretrainedmodels.utils as utils
  model = torch.load('drive/My Drive/public/results/imagenet_nasnetalarge.pth')
  load_img = utils.LoadImage()
  tf_img = utils.TransformImage(model)

import glob
dir_path = './drive/My Drive/public/data/images/'
img_list = glob.glob(dir_path+'*.*')

for img_file in img_list:
  input_img = load_img(img_file)
  input_tensor = tf_img(input_img).to(device) # 3x400x225 -> 3x331x331
  input_tensor = input_tensor.unsqueeze(0)    # 3x331x331 -> 1x3x331x331
  input = torch.autograd.Variable(input_tensor, requires_grad=False)
  output_logits = model(input) # 1x1000

  print("{} is [{}: {}]".format(img_file.split('/')[-1] ,output_logits.argmax(),
                               imagenet_class[int(output_logits.argmax())]))
```

```
test_1.jpg is [282: tiger cat]
cat_224.jpg is [282: tiger cat]
test_0.jpg is [49: African crocodile, Nile crocodile, Crocodylus niloticus]
```

Practice: classify your own image

- take a photo
- put the <your_image> photo in the ‘public/data/images/’ directory
- img_file = './drive/My Drive/public/data/images/<your_image>'
- classify them

Transfer Learning

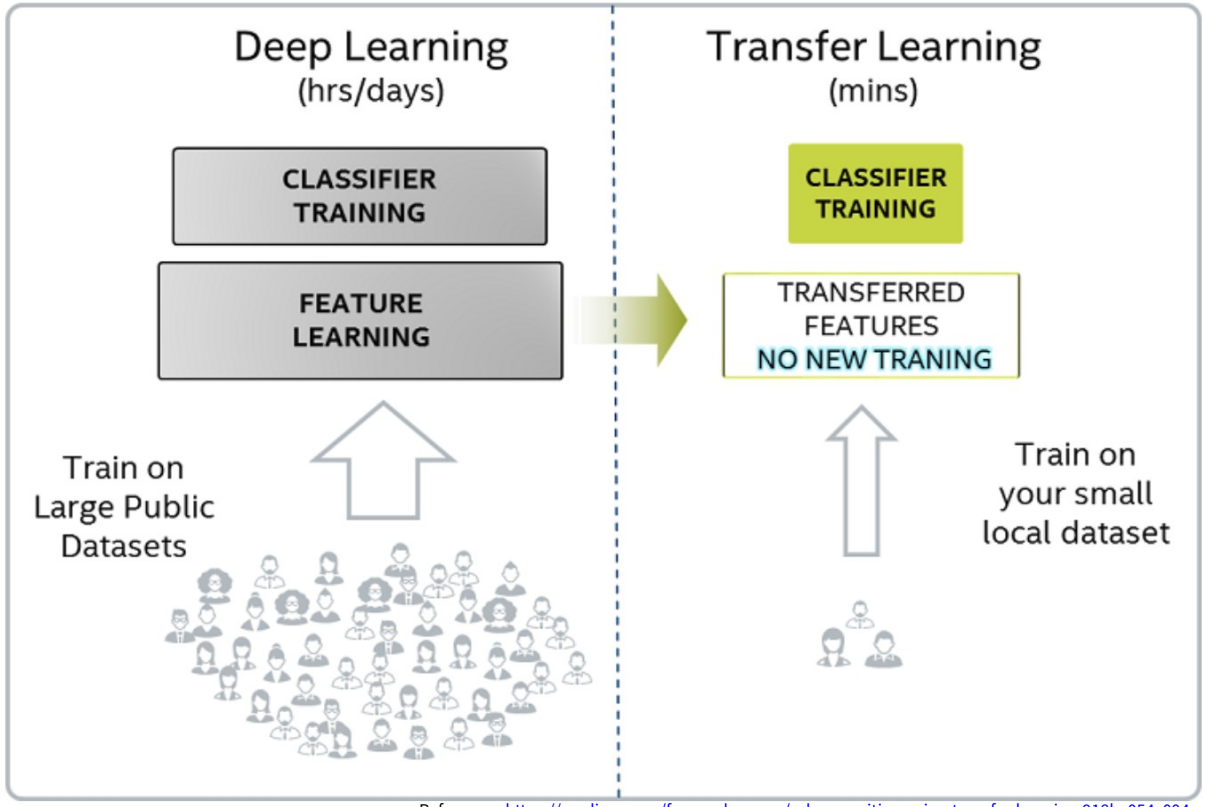
- We don't have a huge dataset
- Some imagenet images look similar with our images
- Pretrained models on imagenet can be useful for training our model

Our data, just 3k

Imagenet, 1.2Million

Transfer Learning

- Using pre-trained model for training
- Re-training the model using our own data



Reference : https://medium.com/free-code-camp/asl-recognition-using-transfer-learning-918ba054c004

Transfer Learning

(After running imagenet example code…)

- Let's see pre-trained model's architecture

```
print(model) # pretrained NASNetALarge which trained by imagenet data
```

```
NASNetALarge(
  (conv0): Sequential(
    (conv): Conv2d(3, 96, kernel_size=(3, 3), stride=(2, 2), bias=False)
    (bn): BatchNorm2d(96, eps=0.001, momentum=0.1, affine=True, track_running_stats=Tru
  )
                                                    :
    (relu): ReLU()                                  :
    (avg_pool): AvgPool2d(kernel_size=11, stride=1, padding=0)
    (dropout): Dropout(p=0.5, inplace=False)
    (last_linear): Linear(in_features=4032, out_features=1000, bias=True)
  )
```

- We want to use pre-trained model layers excepting 'last_linear'
  - Our data have 5 classes, not 1k like imagenet

Transfer Learning

- Re-define model
  - Make pre-trained model architecture and change 'last_linear' layer

```
from pretrainedmodels.models.nasnet import NASNetALarge
output_dim = 5

class TransferedModel(NASNetALarge) :
  def __init__(self):
    super(TransferedModel, self).__init__()

    self.last_linear = nn.Linear(4032, 5)
```

Transfer Learning

- Loading partial parameters from pre-trained model

```
import torch.nn as nn

pretrained_dict = model.state_dict()

model = TransferedModel().to(device)
model_dict = model.state_dict()

## load pretrained model's parameter
# 1. filter out unnecessary keys
pretrained_dict = {k: v for k, v in pretrained_dict.items() if not ('last_linear' in k) }
# 2. overwrite entries in the existing state dict
model_dict.update(pretrained_dict)
# 3. load the new state dict
model.load_state_dict(model_dict)


learning_rate = 0.0005
loss_func = nn.CrossEntropyLoss()

## optimizer without pretrained parameters
for name, param in model.named_parameters() :
  if 'last_linear' in name :
    param.requires_grad = True
  else :
    param.requires_grad = False
params = filter(lambda p: p.requires_grad, model.parameters())
optimizer = torch.optim.Adam(params, lr=learning_rate)
```

Transfer Learning

- Other things are the same as the previous slides
  - Data transform function

```
from torchvision import transforms

resize=(400, 400)
input_size=(331, 331)

data_transforms = {
    'train': transforms.Compose([
        transforms.Resize(resize),
        transforms.RandomCrop(input_size), # data augmentation
        transforms.ToTensor(),
        transforms.Normalize([0.5, 0.5, 0.5], [0.3, 0.3, 0.3])
    ]),
    'valid': transforms.Compose([
        transforms.Resize(resize),
        transforms.CenterCrop(input_size),
        transforms.ToTensor(),
        transforms.Normalize([0.5, 0.5, 0.5], [0.3, 0.3, 0.3])
    ]),
}
test_transform = data_transforms['valid']
```

Transfer Learning

- Other things are the same as the previous slides
  - Data loader

```
from torchvision import datasets

batch_size = 64 # try different batch_size values
data_dir = './drive/My Drive/public/data/'
train_dir = 'train'
valid_dir = 'valid'

train_set = datasets.ImageFolder(data_dir+train_dir, data_transforms['train'])
valid_set = datasets.ImageFolder(data_dir+valid_dir, data_transforms['valid'])

train_loader = torch.utils.data.DataLoader(train_set, batch_size=batch_size,
                                           shuffle=True, num_workers=4)
valid_loader = torch.utils.data.DataLoader(valid_set, batch_size=batch_size,
                                           shuffle=True, num_workers=4)

train_size = len(train_set)
valid_size = len(valid_set)

class_names = train_set.classes

print(class_names)
print(f'Train image size: {train_size}')
print(f'Validation image size: {valid_size}')
```

Transfer Learning

- Other things are the same as the previous slides
  - training

```
import os

result_dir = 'drive/My Drive/public/results/'
num_epoch = 3 # try with different epochs and find the best epoch

if not os.path.exists(result_dir):
    os.mkdir(result_dir)

for i in range(num_epoch):
    model.train()
    for j, [image,label] in enumerate(train_loader):
        x = image.to(device)
        y_= label.to(device)

        optimizer.zero_grad()
        output = model(x)
        loss = loss_func(output,y_)
        loss.backward()
        optimizer.step()

        if j % 30 == 0:
            print(i,j, loss.data.cpu())

    model.eval()
    hits = 0
    for k, [image,label] in enumerate(valid_loader):
        x = image.to(device)
        y_= label.to(device)

        output = model(x)
        y_est = output.argmax(1)
        hits = hits + sum(y_est == y_).cpu()
    print('Hits', int(hits), 'Accuracy', float(hits/(valid_size+0.0)))

torch.save(model, result_dir + 'teamX.model')
print('training is done by max_epochs', num_epoch)
```

```
0 0 tensor(1.6431)
0 30 tensor(0.5285)
0 60 tensor(0.2411)
Hits 229 Accuracy 0.9502074718475342
1 0 tensor(0.2502)
1 30 tensor(0.1757)
1 60 tensor(0.0823)
Hits 234 Accuracy 0.9709543585777283
2 0 tensor(0.1285)
2 30 tensor(0.0996)
2 60 tensor(0.0483)
Hits 236 Accuracy 0.9792531132698059
training is done by max_epochs 3
```

Transfer Learning

- Other things are the same as the previous slides
  - testing

```
test_batch_size = 10
result_dir = 'drive/My Drive/public/results/'
model_name = 'teamX.model'

model = torch.load(result_dir + model_name)
model.to(device)
model.eval()

test_dir = './drive/My Drive/public/data/valid2'
test_set = datasets.ImageFolder(test_dir, test_transform)

test_loader = torch.utils.data.DataLoader(test_set, batch_size=test_batch_size,
                                          shuffle=False, num_workers=4)

hits = 0
for k,[image,label] in enumerate(test_loader):
    x = image.to(device)
    y_= label.to(device)

    output = model(x)
    y_est = output.argmax(1)
    print('Target', label.numpy(), 'Prediction', y_est.cpu().numpy())
    hits = hits + sum(y_est == y_)
print('hits', int(hits),'accuracy', float(hits/(len(test_set)+0.0)))
```

```
Target [0 0 0 0 0 0 0 0 0 0] Prediction [0
Target [0 0 0 0 0 0 0 0 0 0] Prediction [0
Target [0 0 0 0 0 0 0 0 0 0] Prediction [0
Target [0 0 0 0 0 0 0 0 0 0] Prediction [0
Target [1 1 1 1 1 1 1 1 1 1] Prediction [1
Target [1 1 1 1 1 1 1 1 1 1] Prediction [1
Target [1 1 1 1 1 1 1 1 1 1] Prediction [1
Target [1 1 1 1 1 1 1 1 1 1] Prediction [1
Target [2 2 2 2 2 2 2 2 2 2] Prediction [2
Target [2 2 2 2 2 2 2 2 2 2] Prediction [2
Target [2 2 2 2 2 2 2 2 2 2] Prediction [2
Target [2 2 2 2 2 2 2 2 2 2] Prediction [2 2 2 2 2 2 2 2 2 2]
Target [3 3 3 3 3 3 3 3 3 3] Prediction [3 3 3 3 3 3 3 3 4 3]
Target [3 3 3 3 3 3 3 3 3 3] Prediction [3 3 3 3 3 3 3 3 3 3]
Target [3 3 3 3 3 3 3 3 3 3] Prediction [3 3 3 3 3 3 3 3 3 3]
Target [3 3 3 3 3 3 3 3 3 3] Prediction [3 3 3 3 3 3 3 3 3 3]
Target [4 4 4 4 4 4 4 4 4 4] Prediction [4 4 4 4 4 4 4 4 4 4]
Target [4 4 4 4 4 4 4 4 4 4] Prediction [4 4 4 4 4 4 4 4 4 4]
Target [4 4 4 4 4 4 4 4 4 4] Prediction [4 4 4 4 4 4 4 4 4 4]
Target [4 4 4 4 4 4 4 4 4 4] Prediction [2 3 4 3 3 3 4 4 3 4]
hits 193 accuracy 0.9649999737739563
```

Transfer Learning

- Other things are the same as the previous slides
  - testing

```
D04_OH18.jpeg ==> 4 OH
D15_OH31.jpg ==> 4 OH
D03_OH13.jpg ==> 4 OH
D16_OH34.jpg ==> 4 OH
D11_Hyoam05.JPG ==> 2 Hyoam
D09_Hyoam31.JPG ==> 2 Hyoam
D12_OH21.jpg ==> 4 OH
D18_NTH30.jpg ==> 3 NTH
D06_OH29.jpg ==> 4 OH
D08_OH29.jpg ==> 4 OH
D15_OH43.jpg ==> 4 OH
D01_OH22.jpg ==> 4 OH
D11_NTH30.jpeg ==> 3 NTH
D09_NTH30.jpg ==> 3 NTH
D12_NTH15.jpg ==> 3 NTH
D05_NTH31.jpg ==> 3 NTH
D13_ANH45.jpg ==> 0 ANH
D15_Hyoam47.jpg ==> 2 Hyoam
D18_Hyoam40.JPG ==> 2 Hyoam
D12_HDH10.jpg ==> 1 HDH
D06_HDH08.jpg ==> 1 HDH
D07_HDH49.jpg ==> 1 HDH
D17_Hyoam39.jpeg ==> 2 Hyoam
D09_Hyoam09.JPG ==> 2 Hyoam
D01_Hyoam41.jpg ==> 2 Hyoam
D08_Hyoam09.jpg ==> 2 Hyoam
D09_HDH31.JPG ==> 1 HDH
D14_Hyoam12.jpg ==> 2 Hyoam
D10_ANH46.jpg ==> 0 ANH
D07_HDH30.jpg ==> 1 HDH
```

```python
from skimage import io

img_name = './drive/My Drive/public/data/test/test1.jpg'
test_img = io.imread(img_name)
test_img = transforms.ToPILImage()(test_img)
test_img = test_transform(test_img)
test_data = test_img.reshape(1,3,input_size[0],input_size[1]).to(device)
with torch.no_grad():
  output=model(test_data)

class_id = output.argmax(dim=1).cpu().numpy()[0]
print(img_name.split('/')[-1], '==>', class_id, class_names[class_id])
```

```python
from skimage import io
import glob

img_dir = 'drive/My Drive/public/data/test/'
file_list = glob.glob(img_dir + '*.*')
for img_name in file_list:
  test_img = io.imread(img_name)
  test_img = transforms.ToPILImage()(test_img)
  test_img = test_transform(test_img)
  test_data = test_img.reshape(1,3,input_size[0],input_size[1]).to(device)
  with torch.no_grad():
    output=model(test_data)

  class_id = output.argmax(dim=1).cpu().numpy()[0]
  print(img_name.split('/')[-1], '==>', class_id, class_names[class_id])
```

Machine Intelligence Lab
https://milab.handong.edu/

Handong Global University