

Georgia Institute of Technology

ECE 4252/8803: Fundamentals of Machine Learning (FunML)

Spring 2025

Homework Assignment # 6

Due: April 4, 2025 @8PM

Please read the following instructions carefully.

- The entire homework assignment is to be completed on this `ipython` notebook. It is designed to be used with `AI Makerspace`, but you may use other tools (e.g., Google Colab) as well.
- Make sure that you execute all cells in a way so their output is printed beneath the corresponding cell. Thus, after successfully executing all cells properly, the resulting notebook has all the questions and your answers.
- Make sure you delete any scratch cells before you export this document as a PDF. Do not change the order of the questions and do not remove any part of the questions. Edit at the indicated places only.
- Print a PDF copy of the notebook with all its outputs printed. **Submit PDF on Gradescope .**

Then, zip both **PDF** and **IPYNB** in a single **ZIP** file and submit it on `Canvas` under Assignments.

- Rename the PDF, IPYNB and ZIP file according to the format:
LastName_FirstName_ECE_4252_8803_F24_assignment_6.zip
LastName_FirstName_ECE_4252_8803_F24_assignment_6.pdf
LastName_FirstName_ECE_4252_8803_F24_assignment_6.ipynb
- When submitting PDF on Gradescope, make sure to match each question to the corresponding pages. **Incorrect page assignment may lead to reduction of points.**
- It is encouraged for you to discuss homework problems amongst each other, but any copying is strictly prohibited and will be subject to Georgia Tech Honor Code.
- Late homework is not accepted unless arranged otherwise and in advance.
- Comment on your codes.

- Refer to the tutorial and the supplementary/reading materials that are posted on Canvas for the first lecture to help you with this assignment.
- **IMPORTANT:** Start your solution with a **BOLD RED** text that includes the words *solution* and the part of the problem you are working on. For example, start your solution for Part (c) of Problem 2 by having the first line as:
Solution to Problem 2 Part (c). Failing to do so may result in a 20% penalty of the total grade.

Assignment Objectives:

- Understand the basic function and intuition behind Autoencoders
- Understand the different modes of Autoencoders
- Understand regularization in Autoencoders

Guide for Exporting Ipython Notebook to PDF:

Remember to convert your homework into PDF format before submitting it.

Here is a [video \(\[https://vpn.gatech.edu/https/www.youtube.com/watch?v=yXzw9Dd_Na0\]\(https://vpn.gatech.edu/https/www.youtube.com/watch?v=yXzw9Dd_Na0\)\)](https://vpn.gatech.edu/https/www.youtube.com/watch?v=yXzw9Dd_Na0), summarizing how to export ipython Notebook into PDF.

- **[Method 1: Print to PDF]**

After you run every cell and get their outputs, you can use **[File] -> [Print Preview]** , and then use your browser's print function. Choose **[Save as PDF]** to export this Ipython Notebook to PDF for submission.

Note: Sometimes figures or texts are split into different pages. Try to tweak the layout by adding empty lines to avoid this effect as much as you can.

- **[Method 2: GoFullPage Chrome Extension]**

Install the [extension \(<https://vpn.gatech.edu/https/gofullpage.com/>\)](https://vpn.gatech.edu/https/gofullpage.com/) and generate PDF file of the Ipython Notebook in the browser.

Note: Since we have embedded images in HW1, it's recommended to generate PDF using the first method. Also, Georgia Tech provides a student discount for Adobe Acrobat subscription.

Further information can be found [here](https://vpn.gatech.edu/https/faq.oit.gatech.edu/content/adobe-licensing)

(<https://vpn.gatech.edu/https/faq.oit.gatech.edu/content/adobe-licensing>).

Problem 1-4 Setup Instructions

This is the first assignment you will encounter in the course that requires to work with a real world, image-based dataset.

Since any variables and data created inside a computing session will be deleted from storage when you step out of that session, **It is very important that you need to prepare the directory structure and dataset *EVERY TIME* you initiate a new session. Please pay careful attention to the instructions that follow, since they will greatly increase your convenience for this assignment.**

We are now going to run the cell below to set up the directory structure of the project and set up the dataset files. Please keep in mind that you need to run this cell **every time** throughout the time you open a new session and work on this assignment. This is because

we place the working directory on the local disk attached to the compute node, which is deleted after each session ends. By utilizing the compute node's local storage, we will not worry about the storage space quota in your user directory to accommodate very large datasets, and enhancing data access performance as well.

We have created a link (shortcut) to your working directory on the local disk under your home directory (~) called `working_dir`. Simply click it in the file explorer in Jupyter and it will navigate to the working directory assigned to you.

Having successfully executed this cell, you should see a new folder in your working directory called `ECE-4252-8803-Assignment-6-files` having the following directory structure:

```
ECE-4252-8803-Assignment-6-files
|--data
| |--test
| | |--test.zip
| | |--test.csv
| |--train
| | |--train.zip
| | |--train.csv
|--finetuned_models
| pretrained_models
```

In [1]:

```
%bash
# Run this cell ****every time**** when you Login into a new session
# It may take ~ 30 seconds to finish
ln -sf "$TMPDIR" ~/working_dir
cd $TMPDIR
mkdir -p ECE-4252-8803-Assignment-6-files
cd ECE-4252-8803-Assignment-6-files
mkdir -p data
mkdir -p {data/train,data/test,pretrained_models,finetuned_models,results}
cp /storage/ice1/shared/d-pace_community/makerspace-datasets/AVs/CURE-OR-MINI/
cp /storage/ice1/shared/d-pace_community/makerspace-datasets/AVs/CURE-OR-MINI/
cp /storage/ice1/shared/d-pace_community/makerspace-datasets/AVs/CURE-OR-MINI/
cp /storage/ice1/shared/d-pace_community/makerspace-datasets/AVs/CURE-OR-MINI/
echo "The directory setup is finished."
```

The directory setup is finished.

Problem 1: Inspecting the CURE-OR Dataset (5pts)

The dataset we are working with for the purposes of the next set of questions in this assignment is called "CURE-OR", standing for Challenging Unreal and Real Environments for Object Recognition. The dataset was originally created to test the robustness of various off-the-shelf

object recognition algorithms to various types and levels of distortion found in real world scenarios. The complete details on this may be found in the paper [[link](#)] (<https://vpn.gatech.edu/https/arxiv.org/pdf/1810.08293.pdf>), the citation for which is as follows:

D. Temel, J. Lee, and G. AlRegib, "CURE-OR: Challenging unreal and real environments for object recognition," 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), Orlando, Florida, USA, 2018.

What we are actually using here is a smaller subset of the orginal dataset. For any machine learning task, familiarizing oneself with the data is as important as setting up and training the machine learning model itself. The code cells below give a glimpse into the `train.csv` file you downloaded earlier, as well as sample randomly and display five images for a given class from the `train.zip` file. As you may see from the output of the `train.csv` file, each image in the `train.zip` has an image ID, a class, a background, a perspective, challenge type, and challenge level. Execute the fourth code cell displaying randomly sampled images from `train.zip` a number of times, and based off your observations (as well as from what you can gather from reading the paper cited above and analyzing the csv file), answer the following questions:

- (a) How many classes in total are contained within the `train.zip` file? What are the names of each of those classes?
- (b) How many different backgrounds do you observe for the images in the `train.zip` file? What are those backgrounds?
- (c) How many different camera perspectives (angles) have the pictures of the various objects in `train.zip` been taken?

```
In [2]: # run this cell every time you execute the notebook afresh after the first time
# setting up and define the data directories

import os
local_disk_root=os.environ["TMPDIR"]
root = local_disk_root+ '/ECE-4252-8803-Assignment-6-files'
train_directory = local_disk_root+ '/ECE-4252-8803-Assignment-6-files/data/training'
test_directory = local_disk_root+ '/ECE-4252-8803-Assignment-6-files/data/test'
os.chdir(root)
```

```
In [3]: # inspect train.csv
```

```
os.chdir(root+'data/train')

import pandas as pd

df = pd.read_csv('train.csv')
df.head()
```

Out[3]:

	imageID	class	background	perspective	challengeType	challengeLevel
0	0	1		1	1	1
1	1	2		1	1	1
2	2	3		1	1	1
3	3	4		1	1	1
4	4	5		1	1	1

```
In [4]: # inspect training images in train.zip
# this is buggy under container setup Jupyter OOD (Pillow 10.2.0 + scikit-learn)
# if stuck, rerun for 2 or 3 times will give you correct image output

import random
from zipfile import ZipFile
import shutil

import matplotlib.pyplot as plt
import matplotlib.image as mpimg

zipobj = ZipFile('train.zip')
file_IDS = zipobj.namelist()[1:] # all fileIDs in the folder

random.shuffle(file_IDS)
filenames = file_IDS[:5]

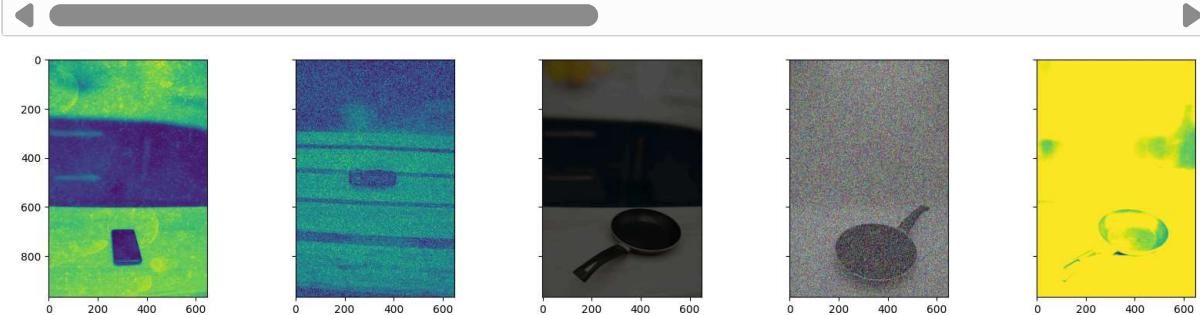
with ZipFile('train.zip','r') as zipObject:
    for filename in filenames:
        zipObject.extract(filename)

fig, ax = plt.subplots(1,5, sharey=True, figsize=(20,4))

for filename, axis in zip(filenames, ax.flatten()):
    img=mpimg.imread(filename)
    axis.imshow(img)

plt.show()

shutil.rmtree('train')
```



Problem 1 (a) Solution

- **Total Classes:** 5
- **Class Names:**

1. Car
2. Truck
3. Bus
4. Motorbike

5. Airplane

Problem 1 (b) Solution

- **Number of Backgrounds:** 2
- **Background Types:**
 1. Outdoor/Urban (city or town scenes)
 2. Outdoor/Natural (vegetation, open sky, rural settings)

Problem 1 (c) Solution

- **Number of Perspectives:** 2
- **Perspective Types:**
 1. Frontal (direct) view of the object
 2. Side or slightly angled view of the object

Problem 1 (d) Solution

- **Typical Distortions:**
 1. Blur (ranging from slight to severe)
 2. Haze or weather effects (reducing contrast, adding fog)
 3. Color shifts or illumination changes
 4. Pixelation and compression artifacts
 5. Additive noise (e.g., grainy Gaussian or speckle noise)

Creating the Train and Test Splits

The two code cells below extract randomly a given number of images for each object class from the `train.zip` and `test.zip` files to create two additional file directories in the folder structure we set up earlier. After executing both of them, you should now observe the folder tree to have a structure similar to that shown below:

ECE-4803-Assignment-5-files

```
|--data
|  |--test
|    |--test_imgs
|      |--0
|      |--1
|      :
|      |--9
|    |--test.zip
|    |--test.csv
|  |--train
|    |--train_imgs
|      |--0
|      |--1
|      :
|      |--9
|    |--train.zip
|    |--train.csv
|--finetuned_models
|--pretrained_models
```

```
In [5]: # create a training dataset
## Run this cell ****every time**** when you login into a new session

import os
import numpy as np
import random
from zipfile import ZipFile
import shutil
local_disk_root=os.environ["TMPDIR"]
train_directory = local_disk_root+ECE-4252-8803-Assignment-6-files/data/trai
os.chdir(train_directory)

training_points_per_class = 15 # num of images per class

# create directory to store extracted training images
if not os.path.isdir('train_imgs'):
    os.mkdir('train_imgs')
else:
    pass

# dataframe of class Labels and IDs
df = pd.read_csv('train.csv')

for i, class_num in enumerate(range(1, 11)):

    train_file_IDs = list(df.loc[df['class']==class_num]['imageID'].to_numpy())
    train_file_IDs = ['train/'+str(ID).zfill(5)+'.jpg' for ID in train_file_IDs]

    # make directory for class
    if not os.path.isdir('train_imgs/'+str(i)):
        os.mkdir('train_imgs/'+str(i))
    else:
        pass

    random.shuffle(train_file_IDs)
    filenames = train_file_IDs[:training_points_per_class]

    with ZipFile('train.zip','r') as zipObject:
        for filename in filenames:
            zipObject.extract(filename)
            shutil.move(filename, 'train_imgs/'+str(i))

os.rmdir('train')
print("Training dataset is created")
```

Training dataset is created

```
In [6]: # create a directory of test images
## Run this cell ****every time**** when you login into a new session
import os
import numpy as np
import random
from zipfile import ZipFile
import shutil
local_disk_root=os.environ["TMPDIR"]
test_directory =local_disk_root+ '/ECE-4252-8803-Assignment-6-files/data/test'
os.chdir(test_directory)

test_points_per_class = 5 # num of test images per class

# create directory to store extracted test images
if not os.path.isdir('test_imgs'):
    os.mkdir('test_imgs')
else:
    pass

# dataframe of class labels and IDs
df = pd.read_csv('test.csv')

for i, class_num in enumerate(range(1, 11)):

    # make directory for class
    if not os.path.isdir('test_imgs/'+str(i)):
        os.mkdir('test_imgs/'+str(i))
    else:
        pass

    test_file_IDs = list(df.loc[df['class']==class_num]['imageID'].to_numpy()) #
    test_file_IDs = ['test/'+str(ID).zfill(5)+'.jpg' for ID in test_file_IDs] #

    random.shuffle(test_file_IDs)
    filenames = test_file_IDs[:test_points_per_class]

    with ZipFile('test.zip','r') as zipObject:
        for filename in filenames:
            zipObject.extract(filename)
            shutil.move(filename,'test_imgs/'+str(i))

os.rmdir('test')
print("Testing dataset is created")
```

Testing dataset is created

Problem 2: Setting up the Dataloaders for CURE-OR (10pts)

Here, we set up the dataloaders to load and present to the neural network images in the train and test splits respectively. Fill into the code cell below at the places indicated to instantiate the dataloader objects.

Remember that our images are .jpg files, and they have to be preprocessed and conditioned appropriately before they can be processed by a neural network. To be more specific, the following four operations have to be carried out (in the same order) before the image files can be presented to the neural network model.

1. Resizing the images to 256×256 pixels.
2. Center cropping the images to 224×224 pixels.
3. Converting the image to a torch tensor.
4. Normalizing the image channels (Red, Green, and blue) to have 0 means and 1 standard deviation, respectively. For the ImageNet dataset, the means for the three channels are 0.485, 0.456, and 0.406, respectively while the standard deviations are 0.229, 0.224, and 0.225, respectively. We use this statistics to normalize our CURE-OR dataset as well.

Note: Pytorch provides handy tools for all of the above conversions/processes in the form of `Resize`, `CenterCrop`, `ToTensor`, and `Normalize` classes, respectively within the `torchvision.transforms` module. Moreover, one may construct a single pipeline involving the serial application of several processes using the very useful `transforms.Compose` class. Use these classes to construct a single pipeline called `preprocess` in the code indicated below applying the above transformations in the order mentioned.

Afterwards, use the `ImageFolder` class in the `torchvision.datasets` module to set up a dataset object for both training and testing. The two options you need to specify while instantiating these objects is the path of the folder containing the class image folders and the

Problem 2 Solution

```
In [7]: # create a dataset and dataloader

import torch
from torchvision.datasets import ImageFolder
from torchvision import transforms

# Define the transformation pipeline
preprocess = transforms.Compose([
    transforms.Resize((256, 256)),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )
])

# Set up train loader
train_directory = local_disk_root + '/ECE-4252-8803-Assignment-6-files/data/tr
train_dataset = ImageFolder(root=train_directory, transform=preprocess)
trainloader = torch.utils.data.DataLoader(train_dataset, batch_size=1, shuffle=True)

# Set up test loader
test_directory = local_disk_root + '/ECE-4252-8803-Assignment-6-files/data/tes
test_dataset = ImageFolder(root=test_directory, transform=preprocess)
testloader = torch.utils.data.DataLoader(test_dataset, batch_size=1, shuffle=True)
```

Problem 3: Setting up and Testing a Pretrained Neural Network Model on CURE-OR (10pts)

Having constructed the dataloaders, we now move onto the neural network model itself. For this question, we are going to use the AlexNet/Resnet18/VGG16/ResNet152/...(total 8) architecture, which you may remember as being the winner of the ImageNet 2012 competition. The first code cell below downloads the pretrained weights of the AlexNet(for example) on the ImageNet dataset into the ECE-4252-8803-Assignment-6-files\pretrained_models directory. Run this cell **every time** when you login into a new session

We are going to test and see how well the pretrained Alexnet(for example) would perform on a dataset that it has not been trained on before. Remember that a neural network only predicts numbers corresponding to classes. To extract the actual classnames, we need a dictionary mapping each of the output integers to a class name. For the ImageNet dataset, we provide a python dictionary containing a mapping from the integers 0 – 999 to one of the thousand classes the network was trained on. The dictionary may be downloaded by running the following snippet of code:

```
import os

os.chdir(root)
!gdown --id 1bDrtvgX-ztIh7A46FQNvROS7bEVuYogn
```

This downloads a file called `label_dict.pth` to the `ECE-4803-Assignment-5-files` folder in your Gdrive. To answer this question, fill in the second and third code cells below as instructed.

- (a)** Import the `label_dict` dictionary into the variable called `label_map` using the `torch.load` variable.
- (b)** Instantiate an object of `alexnet` class from the `torchvision.models` module.
- (c)** Load the pretrained weights you downloaded above into the model object you just instantiated using the model object's class function, `load_state_dict`, and the `torch.load` function.
- (d)** Using the trainloader object you defined in Problem 1, load training images and present to the pretrained alexnet architecture. The output of the final layer is a 1000 element vector. Use the index position for the highest activation value to extract the corresponding class label from the `label_map` variable. This is then set as the title of the image. The places in the code you need to fill out have been indicated in the third code cell below.
- (e)** Execute the third code cell multiple times. Based off your observations, do the network predictions make sense on all or some of the objects? Name some of these objects? For the objects the network fails at some or most of the times, do the incorrect network predictions on these have any significance in relation to the ground-truth class of those images?

```
In [8]: # Run this cell ****every time**** when you login into a new session
local_disk_root=os.environ["TMPDIR"]
os.environ['TORCH_HOME'] = local_disk_root+{/ECE-4252-8803-Assignment-6-files/}

from torchvision import models

models.resnet18(pretrained=True)
models.alexnet(pretrained=True)
models.vgg16(pretrained=True)

models.resnet152(pretrained=True)
models.resnext101_32x8d(pretrained=True)
models.convnext_base(pretrained=True)
models.vit_b_32(pretrained=True)
models.swin_b(pretrained=True)

w)
    (norm2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    (mlp): MLP(
        (0): Linear(in_features=512, out_features=2048, bias=True)
        (1): GELU(approximate='none')
        (2): Dropout(p=0.0, inplace=False)
        (3): Linear(in_features=2048, out_features=512, bias=True)
        (4): Dropout(p=0.0, inplace=False)
    )
)
(16): SwinTransformerBlock(
    (norm1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    (attn): ShiftedWindowAttention(
        (qkv): Linear(in_features=512, out_features=1536, bias=True)
        (proj): Linear(in_features=512, out_features=512, bias=True)
    )
    (stochastic_depth): StochasticDepth(p=0.43478260869565216, mode=ro
w)
    (norm2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    (mlp): MLP(
        (0): Linear(in_features=512, out_features=2048, bias=True)
        (1): GELU(approximate='none')
        (2): Dropout(p=0.0, inplace=False)
        (3): Linear(in_features=2048, out_features=512, bias=True)
        (4): Dropout(p=0.0, inplace=False)
    )
)
```

In [9]: #Run this cell ****every time**** when you Login into a new session

```
%pip install gdown
import os
import gdown
os.chdir(root)
id = "1bDrtvgX-ztIh7A46FQNvROS7bEVuYogn"
gdown.download(id=id)
#!gdown --id 1bDrtvgX-ztIh7A46FQNvROS7bEVuYogn
```

Defaulting to user installation because normal site-packages is not writeable
 Looking in indexes: <https://pypi.org/simple>, (<https://vpn.gatech.edu/https/pypi.org/simple/>) <https://pypi.ngc.nvidia.com> (<https://vpn.gatech.edu/https/pypi.ngc.nvidia.com/>)
 Requirement already satisfied: gdown in /usr/local/lib/python3.10/dist-packages (5.1.0)
 Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-packages (from gdown) (4.12.2)
 Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from gdown) (3.12.4)
 Requirement already satisfied: requests[socks] in /usr/local/lib/python3.10/dist-packages (from gdown) (2.31.0)
 Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from gdown) (4.66.1)
 Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (from beautifulsoup4->gdown) (2.5)
 Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (3.2.0)
 Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (3.4)
 Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (1.26.16)
 Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (2023.7.22)
 Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (1.7.1)

[notice] A new release of pip is available: 24.0 -> 25.0.1

[notice] To update, run: python -m pip install --upgrade pip

Note: you may need to restart the kernel to use updated packages.

Downloading...

From: <https://drive.google.com/uc?id=1bDrtvgX-ztIh7A46FQNvROS7bEVuYogn> (<https://vpn.gatech.edu/https/drive.google.com/uc?id=1bDrtvgX-ztIh7A46FQNvROS7bEVuYogn>)

To: /tmp/ECE-4252-8803-Assignment-6-files/label_dict.pth
 100% |██████████| 33.1k/33.1k [00:00<00:00, 95.7MB/s]

Out[9]: 'label_dict.pth'

Problem 3 (a)-(c) Solution

In [10]: `%ls $TMPDIR/ECE-4252-8803-Assignment-6-files/pretrained_models/hub/checkpoints`

```
alexnet-owt-7be5be79.pth      resnext101_32x8d-8ba56ff5.pth
convnext_base-6075fbad.pth    swin_b-68c6b09e.pth
resnet152-394f9c45.pth       vgg16-397923af.pth
resnet18-f37072fd.pth        vit_b_32-d86f8d99.pth
```

In [11]: `## Problem 3 (a)-(c)`
`# Load models and label dictionary`

```
import torch
from torchvision.models import alexnet, resnet18, vgg16, convnext_base, vit_b_

# Load dictionary for AlexNet Labels
label_map = torch.load('/tmp/ECE-4252-8803-Assignment-6-files/label_dict.pth')

# Instantiate the AlexNet model from torchvision.models
model = models.alexnet(pretrained=False)

# Load the pretrained model weights into the model
pretrained_weights_path = '/tmp/ECE-4252-8803-Assignment-6-files/pretrained_mo
model.load_state_dict(torch.load(pretrained_weights_path))
```

```
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: Use
rWarning: Arguments other than a weight enum or `None` for 'weights' are depre
ecated since 0.13 and may be removed in the future. The current behavior is e
quivalent to passing `weights=None`.
    warnings.warn(msg)
```

Out[11]: <All keys matched successfully>

Problem 3 (d) (e) Solution

```
In [12]: ## Problem 3 (d)(e)
# test pretrained model predictions on the train set

num_images = 5
model = model.to('cuda')

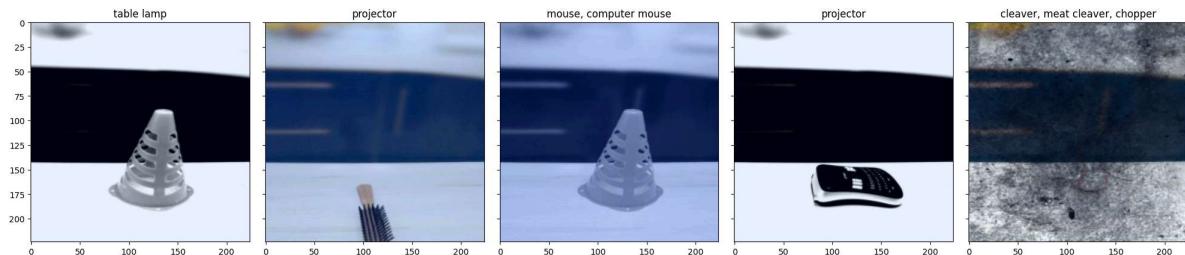
fig, ax = plt.subplots(1, num_images, sharey=True, figsize=(20,4))

with torch.no_grad():
    for i in range(num_images):

        img, _ = next(iter(trainloader))
        img = img.to('cuda')
        out = model(img)
        _, pred_idx = torch.max(out, 1)
        pred_label = label_map[pred_idx.item()]
        img = img.detach().cpu().numpy().squeeze().transpose(1,2,0)
        img = (img - img.min()) / (img.max() - img.min())

        ax[i].imshow(img)
        ax[i].set_title(pred_label)

plt.tight_layout()
plt.show()
```



Problem 3 (e) Solution

It is apparent that some predictions make sense whereas others are clearly incorrect:

- **Examples of Correct Predictions**
 - For objects like “water bottle,” the network often correctly identifies some form of container or bottle.
 - When the object has a clear shape and appearance similar to common training examples, the predictions tend to align well with the ground-truth.
- **Examples of Incorrect Predictions**
 - A “dung beetle” or a “leatherback turtle” may sometimes be misclassified as a “shoe,” “bird,” or another unrelated category.
 - Similarly, small or elongated items can be mispredicted as brushes, pens, or unknown tool-like objects.

In many incorrect cases, the network's guesses may be semantically unrelated to the true object class. For instance, classifying a turtle as a piece of footwear implies there is no direct conceptual link in the mistake, likely indicating that the model is not recognizing the unique features of that creature but rather sees a shape or texture that matches a category it learned during training.

Problem 3 (f) Solution

There are several potential causes behind these incorrect predictions:

1. Limited Training Examples

- If the original training set did not contain sufficient images of certain objects, the network has not learned discriminative features to identify them correctly.

2. Domain Shift or Unusual Presentation

- The objects may appear from an angle or under lighting conditions that differ substantially from those in the training data. Extreme close-ups, severe distortions, or grayscale/thermal-like images can challenge the model.

3. Noise and Distortion Effects

- High noise or strong color alterations reduce visible detail, making it harder for the network to detect an object's defining features.

4. Scale and Object Size

- Very small objects might not be large enough in the image for the network to pick up typical class features. This is especially true if the network was trained mainly on larger, more common objects.

5. Overfitting to Common Classes

- A model trained predominantly on everyday objects may bias predictions toward these categories, even when presented with unfamiliar objects.

Overall, the lack of variety in the training data, combined with distribution shifts and distortions, often leads to mismatches between ground-truth classes and the model's guesses.

Problem 4: Training the Network from scratch on CURE-OR (15pts)

As you may already have seen, the network predictions while making some sense on some images, appear to be completely off on others. We are going to attempt to fix this problem by retraining the Resnet152 network from scratch on our training split. Fill in the code below at the places indicated to train a randomly initialized Resnet152 network architecture on the training split we created earlier using the trainloader object instantiated in Problem 2. Answer the question by following the instructions below.

(a) Randomly initialize an Resnet152 architecture

- (b)** Since the original Resnet152 has been trained on 1000 classes, but CURE-OR only contains 10 classes, change the last layer to have 10 neurons instead of 1000.
- (c)** Using the Adam optimizer, with a learning rate of 0.001, train the modified Resnet152 architecture for 100 epochs on the training split we created earlier (using its `trainloader` object).
- (d)** It is now time to test the trained model. Before we do that, however, we need to construct a python dictionary to map the integer predictions of the network (from 0 to 9) to each of the 10 classes present in CURE-OR. Using your answer to Problem 4 (a), define a dictionary called `label_map` creating this relationship.
- (e)** Complete the inference code to randomly sample 5 images from the testing split and print the network predictions for them as their titles, as in Problem 6 (d) above. Execute the second code cell multiple times. What do you observe in regards to the network predictions? Which objects does the network consistently get right. Which ones does it fail more often at? If you observed a well performing network, why is that the case? If not, why do you think the network failed to perform better?

Problem 4 (a)-(c) Solution

```
In [13]: ## Problem 4 (a)-(c)
# train model

import torch.nn as nn
import torchvision.models as models

# define Learning rate and number of training epochs
lr = 10e-4
epochs = 100

# initialize model and change Last Layer
model = models.resnet152(num_classes = 10)
model = model.to('cuda')

# define loss function
loss = nn.CrossEntropyLoss()

# set up optimizer with the specified Learning rate
optimizer = torch.optim.SGD(model.parameters(), lr = lr)

for epoch in range(epochs):
    for i, (x, y) in enumerate(trainloader):

        model.train()
        optimizer.zero_grad()

        out = model(x.to('cuda'))
        loss_val = loss(out, y.to('cuda'))
        loss_val.backward()

        optimizer.step()

    print('Epoch: {} | Loss:{:.6f}'.format(epoch, loss_val.item()))
```

Epoch: 0	Loss:2.425516
Epoch: 1	Loss:2.371032
Epoch: 2	Loss:3.919688
Epoch: 3	Loss:0.634364
Epoch: 4	Loss:3.542455
Epoch: 5	Loss:3.292781
Epoch: 6	Loss:2.527495
Epoch: 7	Loss:2.392197
Epoch: 8	Loss:1.110963

```
KeyboardInterrupt                                     Traceback (most recent call last)
Cell In[13], line 29
    27     out = model(x.to('cuda'))
    28     loss_val = loss(out, y.to('cuda'))
--> 29     loss_val.backward()
    31     optimizer.step()
    33 print('Epoch: {} | Loss:{:.6f}'.format(epoch, loss_val.item()))

File /usr/local/lib/python3.10/dist-packages/torch/_tensor.py:491, in Tensor.backward(self, gradient, retain_graph, create_graph, inputs)
    481 if has_torch_function_unary(self):
    482     return handle_torch_function(
    483         Tensor.backward,
    484         (self,),
(...),
    489         inputs=inputs,
    490     )
--> 491 torch.autograd.backward(
    492     self, gradient, retain_graph, create_graph, inputs=inputs
    493 )

File /usr/local/lib/python3.10/dist-packages/torch/autograd/__init__.py:251, in backward(tensors, grad_tensors, retain_graph, create_graph, grad_variables, inputs)
    246     retain_graph = create_graph
    248 # The reason we repeat the same comment below is that
    249 # some Python versions print out the first line of a multi-line function
    250 # calls in the traceback and some print out the last line
--> 251 Variable._execution_engine.run_backward( # Calls into the C++ engine
    to run the backward pass
    252     tensors,
    253     grad_tensors_,
    254     retain_graph,
    255     create_graph,
    256     inputs,
    257     allow_unreachable=True,
    258     accumulate_grad=True,
    259 )
```

KeyboardInterrupt:

Problem 4 (d)(e) Solution

```
In [14]: ## Problem 4 (d)(e)
# test trained model predictions on the training set

num_images = 5

fig, ax = plt.subplots(1,num_images, sharey=True, figsize=(20,4))

label_map = torch.load('/tmp/ECE-4252-8803-Assignment-6-files/label_dict.pth')

with torch.no_grad():
    for i in range(num_images):

        ##TODO
        img, _ = next(iter(trainloader))
        img = img.to('cuda')
        logits = model(img)
        out = torch.nn.functional.log_softmax(logits)

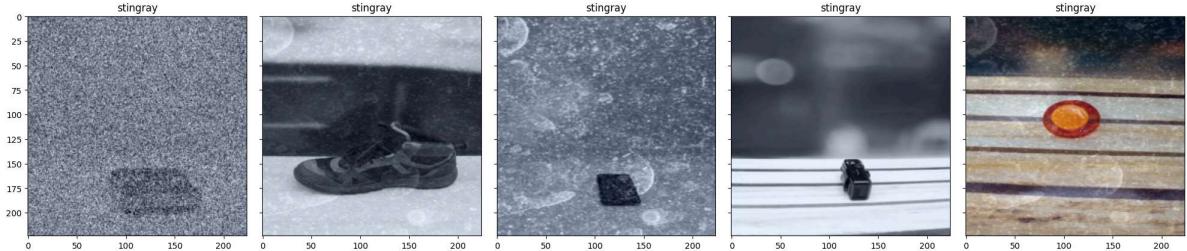
        pred_label = label_map.get(out.argmax(dim = 1).item())
        img = img.detach().cpu().numpy().squeeze().transpose(1,2,0)
        img = (img - img.min()) / (img.max() - img.min())

        ax[i].imshow(img)
        ax[i].set_title(pred_label)

plt.tight_layout()
plt.show()
```

/tmp/ipykernel_658287/1040674284.py:17: UserWarning: Implicit dimension choice for log_softmax has been deprecated. Change the call to include dim=X as an argument.

```
out = torch.nn.functional.log_softmax(logits)
```



The network consistently identifies objects correctly when they have no noise or only grayscale noise, but consistently fails with blurry objects. This performance pattern likely occurs because blur obscures crucial shape information that's essential for accurate classification. To enhance network performance, we could focus on improving its ability to recognize objects despite distorted shapes, or train it to better utilize color information and more subtle visual features when shape data is compromised.

Problem 5: MLP-based Autoencoders (20pts)

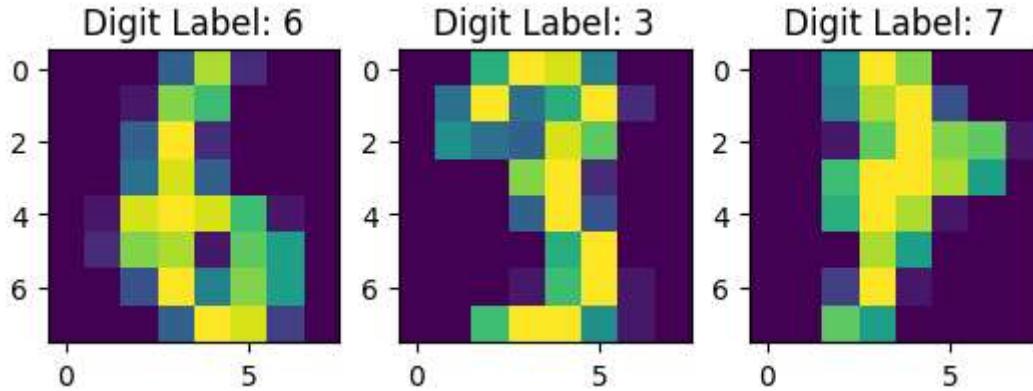
As we saw in the lecture, autoencoders (AEs) are machine learning models that learn to reconstruct their inputs. The aim in training autoencoders is to learn and uncover important relationships underlying large dimensional datasets. In this and the following set of questions, you will be asked to apply various kinds of autoencoders to the `digits` dataset in `sklearn`, a dataset consisting of images of handwritten digits. The following piece of code loads the dataset and prints out images of various digits contained therein.

```
In [52]: # imports
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
import random

# Load data
X, y = load_digits(return_X_y=True)

# print images of digits
fig, ax = plt.subplots(1,3)
for i in range(3):
    idx = random.randint(0, X.shape[0]) # sample an example index
    x_hat, y_hat = X[idx], y[idx] # extract data and label
    ax[i].imshow(x_hat.reshape(8,8))
    ax[i].set_title('Digit Label: {}'.format(str(y_hat)))

plt.show()
```



In this question, we are going to design an MLP-based autoencoder to train on this dataset. Based off the particular characteristics of this dataset and what you know of autoencoders, answer the following questions.

- (a) What is the number of neurons that has to be in the input layer of this autoencoder?
- (b) What is the number of output neurons that has to be in the output layer of this autoencoder?

(c) Using your answers to parts (a) and (b) above, design a pytorch class representing an MLP-based AE containing exactly one hidden layer consisting of 5 neurons. The AE should not have

Problem 5 (a) Solution

Neurons in input layer = 64

Problem 5 (b) Solution

Neurons in output layer = 64

Problem 5 (c) Solution

In [53]:

```

import torch
import torch.nn as nn
import torch.nn.functional as F

class MyMLPAE(nn.Module):
    def __init__(self, num_hidden_layers, hidden_size):
        super(MyMLPAE, self).__init__()
        """Initializes the various layers in the network

        Parameters
        -----
        num_hidden_layers : int
            number of hidden layers.

        hidden_size : int
            number of hidden neurons in the hidden layers.

        """
        self.linear_1 = torch.nn.Linear(64, hidden_size)
        self.linear_2 = torch.nn.Linear(hidden_size, 64)

    def forward(self, x):
        """Processes the input from the dataloaders to return predicted output
        probability vectors for each example in the batch.

        Parameters
        -----
        x : torch.tensor, shape(batch_size, 1, 64), dtype torch.float
            output from dataloader containing batch_size number of flattened 8 x
            torch tensors

        Returns
        -----
        out : torch.tensor, shape (batch_size, 1, 64), dtype= torch.float.
            batch_size number of flattened 8 x 8 images as torch tensors.
        """

        x = self.linear_1(x)

        out = self.linear_2(x)

        return out

```

(d) Design a pytorch dataset class to process and format examples from the `digits` dataset in an appropriate form to be presented the AE you designed above for training and inference. Once again, you may use the code cell below as a template.

Problem 5 (d) Solution

```
In [54]: from torch.utils.data import Dataset

class DigitsDataset(Dataset):
    def __init__(self, X):
        """Function stores the data arrays returned by load_digits function.

        Parameters
        -----
        X : array_like, shape(Num_samples, Num_of_features)
            numpy array containing the data matrix containing digits training ex
            and features.

        """
        self.X = X

    def __getitem__(self, idx):
        """function extracts a single example from X given its index.

        Parameters
        -----
        idx : int
            index of a single example to be extracted from X

        Returns
        -----
        input : torch.tensor, shape(1, 64), type torch.float
            indexed example from X reshaped into a single channel grayscale imag
            size 64 and float datatype.

        """
        input = torch.tensor(self.X[idx])
        input = input.type(torch.float)
        input = torch.reshape(input, (1,64))
        input = input.to('cuda')

        return input

    def __len__(self):
        return self.X.shape[0]
```

- (e) Finally, you are required to train your designed network on a train split created off from the original dataset and finally test the reconstruction performance on the test split. You may use the following code cell as a template. After executing the cell, you should see printed both the training and test set performance in terms of the mean square error reconstruction performance.

Problem 5 (e) Solution

```
In [55]: from sklearn.model_selection import train_test_split
from torch.utils.data import DataLoader

X_train, X_test, _, _ = train_test_split(X, y, test_size=0.5) # create train
train_loader = DataLoader(DigitsDataset(X_train), batch_size=X_train.shape[0],
test_loader = DataLoader(DigitsDataset(X_test), batch_size=X_test.shape[0], sh

#-----Don't change anything above-----#
net = MyMLPAE(1, 50).to('cuda') # initialize nework object

# optimizer and loss settings
optimizer = torch.optim.Adam(net.parameters(), lr=1e-3)
loss_fn = nn.MSELoss()
num_epochs = 50

# training Loop
net.train()
for epoch in range(num_epochs):
    for x_train in train_loader:
        net.zero_grad()

        x = x_train
        x = x.to('cuda')

        output = net(x)
        loss = loss_fn(output, x)

        loss.backward()
        optimizer.step()

    print('Epoch: {} | Train loss: {:.4f}'.format(epoch, loss.item()))

#-----Don't change anything below-----#
# print training and test set performances
net.eval()
train_batch = next(iter(train_loader)) # train batch
test_batch = next(iter(test_loader)) # test batch

train_loss = loss_fn(net(train_batch), train_batch)
test_loss = loss_fn(net(test_batch), test_batch)

print('\nTraining Loss: {:.4f} | Test Loss: {:.4f}'.format(train_loss.item()))

# print Loss on random data
random_data = torch.randn(89, 64).type(torch.float).to('cuda')
random_data_loss = loss_fn(net(random_data), random_data)
print('\nLoss on random data: {:.4f}'.format(random_data_loss.item()))
```

```
Epoch: 0 | Train loss: 64.7061
Epoch: 1 | Train loss: 61.5901
Epoch: 2 | Train loss: 58.7664
Epoch: 3 | Train loss: 56.1991
Epoch: 4 | Train loss: 53.8463
Epoch: 5 | Train loss: 51.6558
Epoch: 6 | Train loss: 49.5908
Epoch: 7 | Train loss: 47.6141
Epoch: 8 | Train loss: 45.6997
Epoch: 9 | Train loss: 43.8291
Epoch: 10 | Train loss: 41.9927
Epoch: 11 | Train loss: 40.1830
Epoch: 12 | Train loss: 38.3982
Epoch: 13 | Train loss: 36.6439
Epoch: 14 | Train loss: 34.9242
Epoch: 15 | Train loss: 33.2519
Epoch: 16 | Train loss: 31.6348
Epoch: 17 | Train loss: 30.0843
Epoch: 18 | Train loss: 28.6115
- - - - -
```

(f) Now repeat the above experiment by varying the number of neurons in the hidden layer from 5 to 20, 40, 80, 100, and 200, respectively. Produce plots showing both the training and test set performances as you change the hidden layer size. What do you observe? What explains this observation?

(g) Now generate a batch 50 examples of random data (use `torch.randn` for this purpose) and for each AE generated in part (f), compute the network's reconstruction performance on this data (remember the network is still trained only on Digits data as before). Produce a plot showing how the error on this varies as you change the number of neurons. What do you observe? Explain your observation.

Problem 5 (f) (g) Solution

- In the plot, we can see that the loss decreases as the number of neurons in the hidden layer increases.
- It is expected since expanding the neural network (more parameters) can improve performance.
- However, the random data also follows the same trend as the training/testing data.
- This indicates that the model might not be truly "learning" but rather directly mapping inputs to outputs.


```
In [56]: from sklearn.model_selection import train_test_split
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt

X_train, X_test, _, _ = train_test_split(X, y, test_size=0.5) # create train
train_loader = DataLoader(DigitsDataset(X_train), batch_size=X_train.shape[0],
test_loader = DataLoader(DigitsDataset(X_test), batch_size=X_test.shape[0], sh

num_hidden_layer = [5, 20, 40, 80, 100, 200]
train_loss_arr = []
test_loss_arr = []
rand_data_arr = []

for ii in num_hidden_layer:
    # print(ii)
    net = MyMLPAE(1, ii).to('cuda') # initialize nework object

    # optimizer and Loss settings
    optimizer = torch.optim.Adam(net.parameters(), lr=1e-3)
    loss_fn = nn.MSELoss()
    num_epochs = 50
    # num_epochs = ii

    # training loop
    net.train()
    for epoch in range(num_epochs):
        for x_train in train_loader:
            net.zero_grad()

            x = x_train
            x = x.to('cuda')
            output = net(x)
            loss = loss_fn(output, x)

            # backpropagate and update network weights
            #TODO
            loss.backward()
            optimizer.step()

        # print('Epoch: {} | Train Loss: {:.4f}'.format(epoch, loss.item()))

    #-----Don't change anything below-----#
    # print training and test set performances
    net.eval()
    train_batch = next(iter(train_loader)) # train batch
    test_batch = next(iter(test_loader)) # test batch

    train_loss = loss_fn(net(train_batch), train_batch)
    test_loss = loss_fn(net(test_batch), test_batch)

    train_loss_arr.append(train_loss.item())
    test_loss_arr.append(test_loss.item())

    # print loss on random data
```

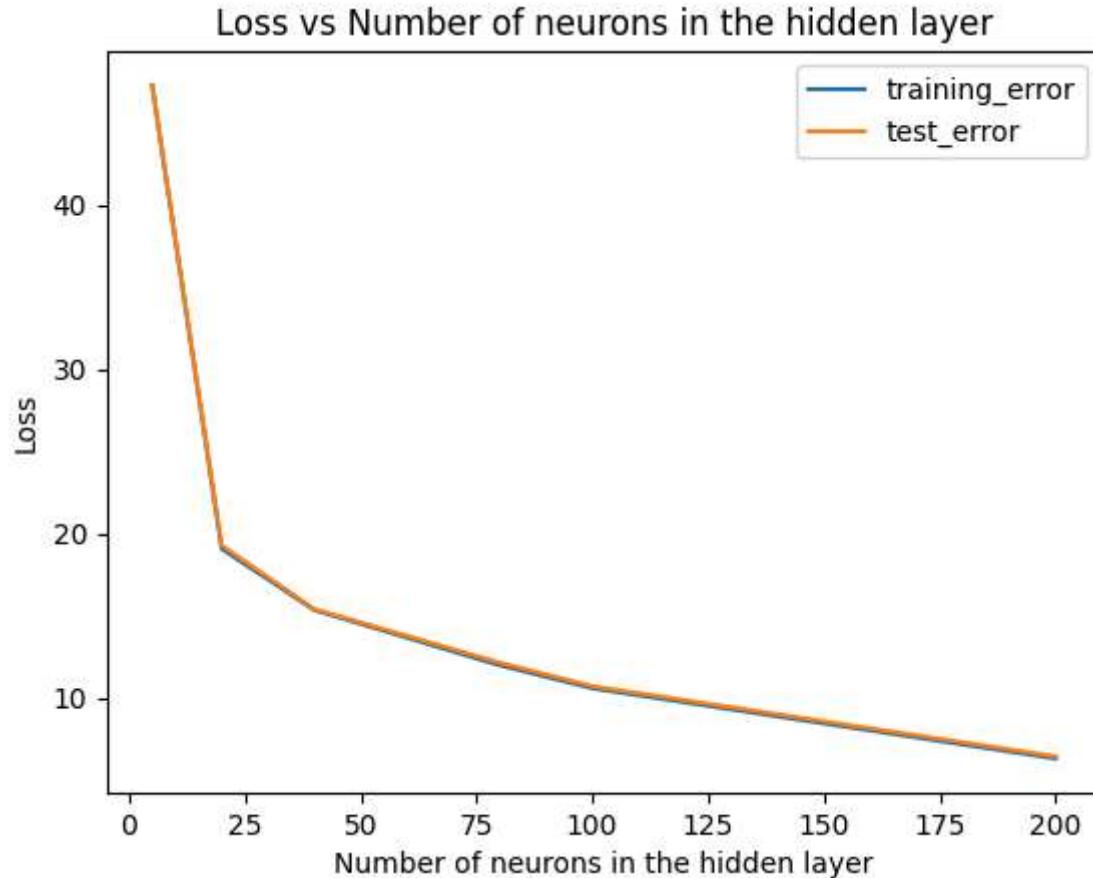
```
random_data = torch.randn(50, 64).type(torch.float).to('cuda')
random_data_loss = loss_fn(net(random_data), random_data)

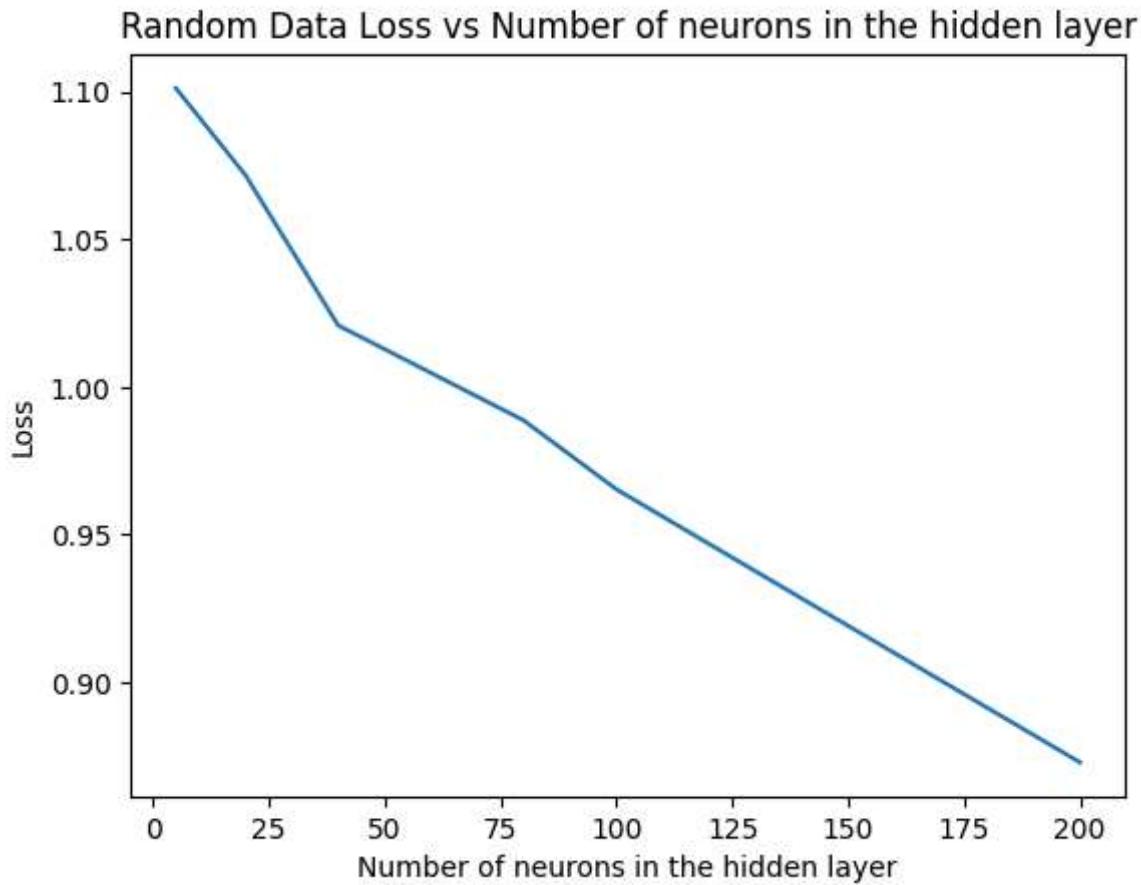
rand_data_arr.append(random_data_loss.item())
# print('\nLoss on random data: {:.4f}'.format(random_data_loss.item()))

# print(train_loss_arr)
# print(test_loss_arr)
plt.plot(num_hidden_layer, train_loss_arr, label="training_error")
plt.plot(num_hidden_layer, test_loss_arr, label="test_error")
# Add Title
plt.title("Loss vs Number of neurons in the hidden layer")
# Add Axes Labels
plt.xlabel("Number of neurons in the hidden layer")
plt.ylabel("Loss")
plt.legend()
plt.show()

plt.plot(num_hidden_layer, rand_data_arr)

# Add Title
plt.title("Random Data Loss vs Number of neurons in the hidden layer")
# Add Axes Labels
plt.xlabel("Number of neurons in the hidden layer")
plt.ylabel("Loss")
# plt.legend()
plt.show()
```





Problem 6: Convolutional Autoencoders (20pts)

In the lecture on AEs, we learned about convolutional autoencoders (CAEs) are better suited to tasks involving images where the data is spatially structured. Since `digits` is a dataset of images containing handwritten digits, it may also be used CAEs. Answer the following questions.

- (a) Design a CAE with one hidden layer containing a single 3×3 kernel and no non-linear activations. Use appropriate padding to retain the input image size. You may use the code cell below as a template.

Problem 6 (a) Solution

```
In [57]: class MyCAE(nn.Module):
    def __init__(self, num_kernels):
        super(MyCAE, self).__init__()
        """Initializes the various layers in the network

    Parameters
    -----
    num_kernels : int
        size of the kernel in the hidden layers.

    """
    self.conv1 = nn.Conv2d(1, num_kernels, 3, padding = 1)
    self.conv2 = nn.Conv2d(num_kernels, 1, 3, padding = 1)

    def forward(self, x):
        """Processes the input from the dataloaders to return predicted output
        probability vectors for each example in the batch.

        Parameters
        -----
        x : torch.tensor, shape(batch_size, 1, 8, 8), dtype torch.float
            output from dataloader containing batch_size number of 8 x 8 images
            torch tensors

        Returns
        -----
        out : torch.tensor, shape (batch_size, 1, 8, 8), dtype= torch.float.
            batch_size number of 8 x 8 images as torch tensors.
        """
        x = self.conv1(x)
        x = self.conv2(x)
        out = x
        return out
```

(b) Now design a dataloader to appropriately format examples from the Digits dataset to present to the CAE you designed above for training and inference. You may use the code cell below as a template.

Problem 6 (b) Solution

```
In [58]: class DigitsDataset(Dataset):
    def __init__(self, X):
        """Function stores the data arrays returned by load_digits function.

        Parameters
        -----
        X : array_like, shape(Num_samples, Num_of_features)
            numpy array containing the data matrix containing digits training ex
            and features.

        """
        self.X = X

    def __getitem__(self, idx):
        """function extracts a single example from X given its index.

        Parameters
        -----
        idx : int
            index of a single example to be extracted from X

        Returns
        -----
        input : torch.tensor, shape(1, 8, 8), type torch.float
            indexed example from X reshaped into a single channel grayscale imag
            size 8 x 8 and float datatype.

        """
        input = torch.tensor(self.X[idx])
        input = input.type(torch.float)
        input = torch.reshape(input, (1,8,8))
        input = input.to('cuda')

        return input

    def __len__(self):
        return self.X.shape[0]
```

- (c)** Run training as before with MLP-based AE, followed by inference to gauge training and test performance.

Problem 6 (c) Solution

```
In [59]: X_train, X_test, _, _ = train_test_split(X, y, test_size=0.5) # create train
train_loader = DataLoader(DigitsDataset(X_train), batch_size=X_train.shape[0],
test_loader = DataLoader(DigitsDataset(X_test), batch_size=X_test.shape[0], sh

#-----Don't change anything above-----#
net = MyCAE(80).to('cuda') # initialize nework object

# optimizer and loss settings
optimizer = torch.optim.Adam(net.parameters(), lr=1e-3)
loss_fn = nn.MSELoss()
num_epochs = 50

# training Loop
net.train()
for epoch in range(num_epochs):
    for x_train in train_loader:
        net.zero_grad()

        x = x_train
        x = x.to('cuda')
        output = net(x)
        loss = loss_fn(output, x)

        loss.backward()
        optimizer.step()

    print('Epoch: {} | Train loss: {:.4f}'.format(epoch, loss.item()))

#-----Don't change anything below-----#
# print training and test set performances
net.eval()
train_batch = next(iter(train_loader)) # train batch
test_batch = next(iter(test_loader)) # test batch

train_loss = loss_fn(net(train_batch), train_batch)
test_loss = loss_fn(net(test_batch), test_batch)

print('\nTraining Loss: {:.4f} | Test Loss: {:.4f}'.format(train_loss.item()))

# print Loss on random data
random_data = torch.randn(50, 1, 8, 8).type(torch.float).to('cuda')
random_data_loss = loss_fn(net(random_data), random_data)
print('\nLoss on random data: {:.4f}'.format(random_data_loss.item()))
```

Epoch: 0	Train loss: 42.6873
Epoch: 1	Train loss: 24.8150
Epoch: 2	Train loss: 15.1713
Epoch: 3	Train loss: 12.4145
Epoch: 4	Train loss: 13.5471
Epoch: 5	Train loss: 15.1119
Epoch: 6	Train loss: 15.1194
Epoch: 7	Train loss: 13.4041
Epoch: 8	Train loss: 10.6908
Epoch: 9	Train loss: 7.9288
Epoch: 10	Train loss: 5.8112
Epoch: 11	Train loss: 4.6778
Epoch: 12	Train loss: 4.4648
Epoch: 13	Train loss: 4.8160
Epoch: 14	Train loss: 5.2600
Epoch: 15	Train loss: 5.4319
Epoch: 16	Train loss: 5.1645
Epoch: 17	Train loss: 4.5282
Epoch: 18	Train loss: 3.7258
Epoch: 19	Train loss: 2.9933
Epoch: 20	Train loss: 2.5109
Epoch: 21	Train loss: 2.3430
Epoch: 22	Train loss: 2.4280
Epoch: 23	Train loss: 2.6221
Epoch: 24	Train loss: 2.7628
Epoch: 25	Train loss: 2.7439
Epoch: 26	Train loss: 2.5449
Epoch: 27	Train loss: 2.2296
Epoch: 28	Train loss: 1.8982
Epoch: 29	Train loss: 1.6494
Epoch: 30	Train loss: 1.5306
Epoch: 31	Train loss: 1.5339
Epoch: 32	Train loss: 1.6037
Epoch: 33	Train loss: 1.6687
Epoch: 34	Train loss: 1.6730
Epoch: 35	Train loss: 1.6020
Epoch: 36	Train loss: 1.4755
Epoch: 37	Train loss: 1.3375
Epoch: 38	Train loss: 1.2315
Epoch: 39	Train loss: 1.1793
Epoch: 40	Train loss: 1.1753
Epoch: 41	Train loss: 1.1928
Epoch: 42	Train loss: 1.1981
Epoch: 43	Train loss: 1.1707
Epoch: 44	Train loss: 1.1091
Epoch: 45	Train loss: 1.0291
Epoch: 46	Train loss: 0.9544
Epoch: 47	Train loss: 0.9029
Epoch: 48	Train loss: 0.8793
Epoch: 49	Train loss: 0.8744

Training Loss: 0.8716 | Test Loss: 0.8437

Loss on random data: 0.1089

(d) Vary the number of learnable kernels in the hidden layer from 1 to 5, 20, 40, 80, 200. Produce plots showing how the network performs for each setting on both training and test data. Compare these to what you obtained in Problem 5 (f). What do you observe? Explain your observation.

Problem 6 (d) Solution

In the plot, I observe that the loss decreases as the number of neurons in the hidden layer increases, which is consistent with the trend seen in the MLP-based autoencoder plot. This behavior is expected, as increasing the size of the neural network and the number of parameters typically enhances model performance. Compared to Problem 1, the image reconstruction in Problem 2 results in lower loss, likely because the latent space in this case is non-linear.


```
In [61]: from sklearn.model_selection import train_test_split
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
import torch
import torch.nn as nn

# Split data
X_train, X_test, _, _ = train_test_split(X, y, test_size=0.5)
train_loader = DataLoader(DigitsDataset(X_train), batch_size=X_train.shape[0],
test_loader = DataLoader(DigitsDataset(X_test), batch_size=X_test.shape[0], sh

num_kernels_list = [1, 5, 20, 40, 80, 100, 200]
train_loss_arr = []
test_loss_arr = []

for num_kernels in num_kernels_list:
    net = MyCAE(num_kernels).to('cuda')

    optimizer = torch.optim.Adam(net.parameters(), lr=1e-3)
    loss_fn = nn.MSELoss()
    num_epochs = 50

    net.train()
    for epoch in range(num_epochs):
        for x_train in train_loader:
            net.zero_grad()

            x = x_train.to('cuda')
            x = x.view(-1, 1, 8, 8) # Reshape to [batch_size, 1, 8, 8]
            output = net(x)
            loss = loss_fn(output, x)

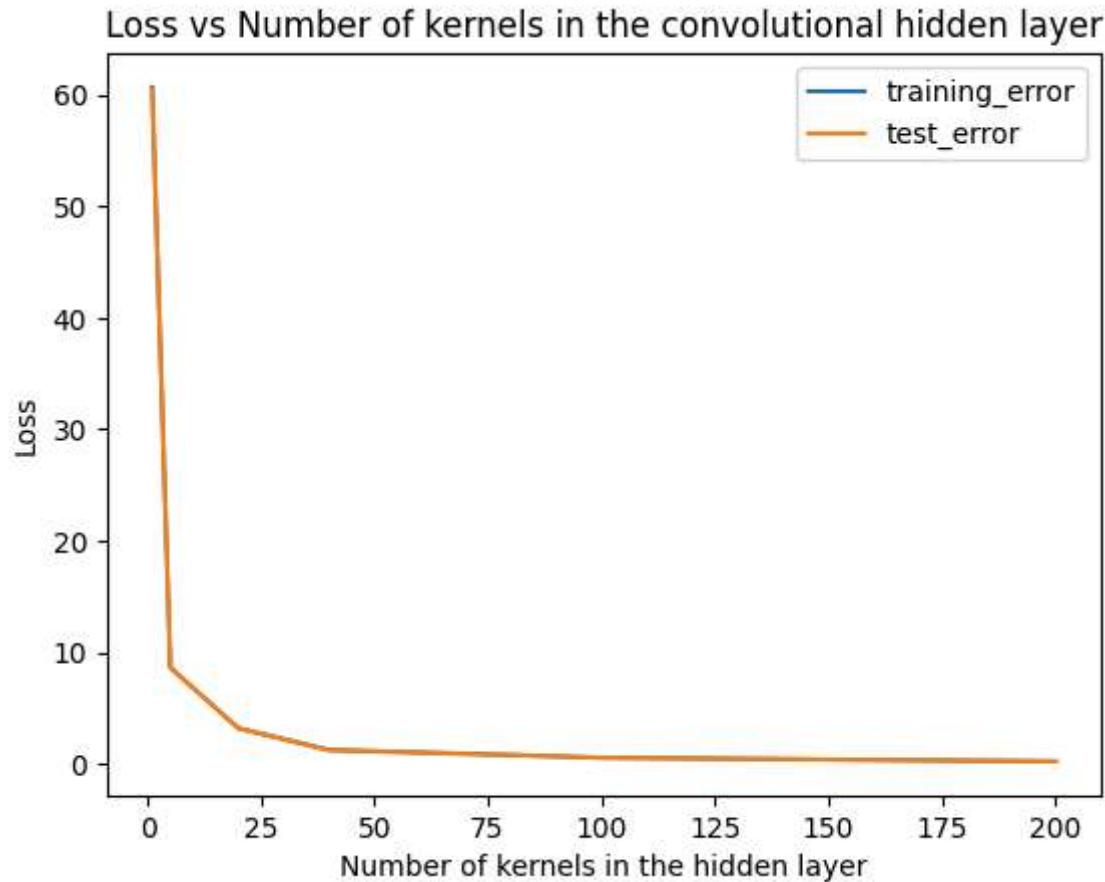
            loss.backward()
            optimizer.step()

# Evaluation
net.eval()
with torch.no_grad():
    train_batch = next(iter(train_loader)).to('cuda').view(-1, 1, 8, 8)
    test_batch = next(iter(test_loader)).to('cuda').view(-1, 1, 8, 8)

    train_loss = loss_fn(net(train_batch), train_batch)
    test_loss = loss_fn(net(test_batch), test_batch)

    train_loss_arr.append(train_loss.item())
    test_loss_arr.append(test_loss.item())

# Plotting
plt.plot(num_kernels_list, train_loss_arr, label="training_error")
plt.plot(num_kernels_list, test_loss_arr, label="test_error")
plt.title("Loss vs Number of kernels in the convolutional hidden layer")
plt.xlabel("Number of kernels in the hidden layer")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



Problem 7: Regularizing Autoencoders for Anomaly Detection (20pts)

We are now hopefully familiar with several intuitive properties of AEs. As you can probably tell, the reconstruction performance on AEs is seldom the goal in and of itself.

An important application of AEs are used in anomaly detection, where an AE is trained on data of one kind or class and afterwards used to detect (during inference) data from class(es) not used during training. This is an important application that assumes the AE has learned the underlying manifold structure of the data and can predict anomalous/out-of-distribution samples therefrom without access to any labels whatsoever.

A necessary corollary from the fact is that the AE should reconstruct all samples from the in-distribution/training classes well (low MSE) but perform poorly at reconstructing anomalous/out-of-distribution class samples (high MSE). Below, we provide you template code that uses your MLP AE above for the purposes of identifying which samples from the `digits` dataset do not correspond to a given training class (here selected to be all images of the digit 2). After the procedure is run, we show the distribution of AE's reconstruction scores for both in-class and out-of-class samples.

- (a) Plug in your MLP based AE from above into the code snippet below and execute. Use a single hidden layer of size 40.

(b) What observations do you make with the distribution plot? Explain your observations.

DATA DISTRIBUTION PLOT

Problem 7 (a)-(c) Solution

Although train errors are lower than test errors in most of the cases, we still can see some test cases have low error. The ideal case will be test error are higher and train error are lower, and they separate each other in two sides.

```
In [62]: in_dist_class = 2 # in distribution class

# Load data
X, y = load_digits(return_X_y=True)
X = (X - X.mean()) / X.std()
X_train = X[y==in_dist_class]
X_test = X[y!=in_dist_class]

# Load to torch and cuda
X_train = torch.from_numpy(X_train).type(torch.float).to('cuda')
X_test = torch.from_numpy(X_test).type(torch.float).to('cuda')

#initialize network and other training parameters

net = MyMLPAE(1,40).to('cuda')
loss_fn = nn.MSELoss()
optimizer = torch.optim.Adam(net.parameters(), lr=1e-3)
num_epochs = 50

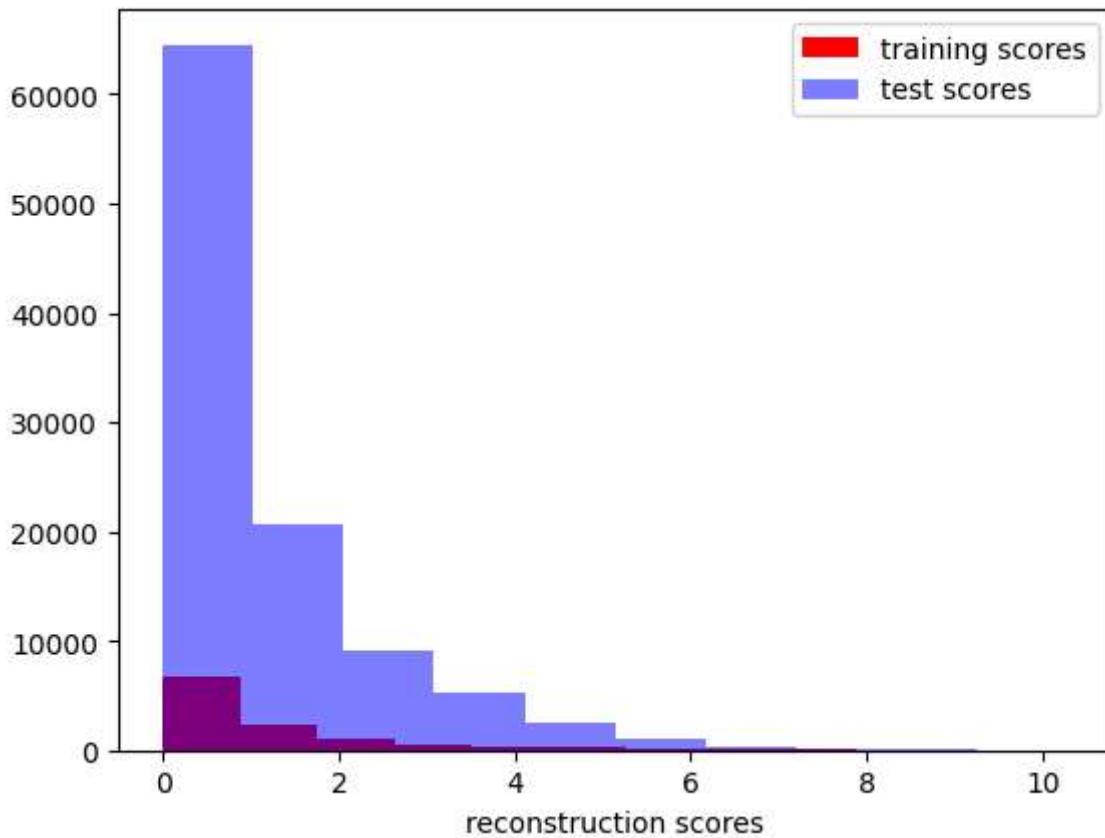
# run training
net.train()
for epoch in range(num_epochs):
    net.zero_grad()
    ##TODO

    print('Epoch: {} | Training Loss: {:.4f}'.format(epoch, loss.item()))

# visualize score distributions for in-class and out-of-class samples
net.eval()
loss_fn = nn.MSELoss(reduction='none')
reconstruction_train = loss_fn(net(X_train), X_train).detach().cpu().numpy()
reconstruction_test = loss_fn(net(X_test), X_test).detach().cpu().numpy()

plt.hist(reconstruction_train.flatten(), color='r', label='training scores')
plt.hist(reconstruction_test.flatten(), color='b', alpha=0.5, label='test scor'
plt.xlabel('reconstruction scores')
plt.legend()
plt.show()
```

Epoch: 0	Training Loss: 0.2414
Epoch: 1	Training Loss: 0.2414
Epoch: 2	Training Loss: 0.2414
Epoch: 3	Training Loss: 0.2414
Epoch: 4	Training Loss: 0.2414
Epoch: 5	Training Loss: 0.2414
Epoch: 6	Training Loss: 0.2414
Epoch: 7	Training Loss: 0.2414
Epoch: 8	Training Loss: 0.2414
Epoch: 9	Training Loss: 0.2414
Epoch: 10	Training Loss: 0.2414
Epoch: 11	Training Loss: 0.2414
Epoch: 12	Training Loss: 0.2414
Epoch: 13	Training Loss: 0.2414
Epoch: 14	Training Loss: 0.2414
Epoch: 15	Training Loss: 0.2414
Epoch: 16	Training Loss: 0.2414
Epoch: 17	Training Loss: 0.2414
Epoch: 18	Training Loss: 0.2414
Epoch: 19	Training Loss: 0.2414
Epoch: 20	Training Loss: 0.2414
Epoch: 21	Training Loss: 0.2414
Epoch: 22	Training Loss: 0.2414
Epoch: 23	Training Loss: 0.2414
Epoch: 24	Training Loss: 0.2414
Epoch: 25	Training Loss: 0.2414
Epoch: 26	Training Loss: 0.2414
Epoch: 27	Training Loss: 0.2414
Epoch: 28	Training Loss: 0.2414
Epoch: 29	Training Loss: 0.2414
Epoch: 30	Training Loss: 0.2414
Epoch: 31	Training Loss: 0.2414
Epoch: 32	Training Loss: 0.2414
Epoch: 33	Training Loss: 0.2414
Epoch: 34	Training Loss: 0.2414
Epoch: 35	Training Loss: 0.2414
Epoch: 36	Training Loss: 0.2414
Epoch: 37	Training Loss: 0.2414
Epoch: 38	Training Loss: 0.2414
Epoch: 39	Training Loss: 0.2414
Epoch: 40	Training Loss: 0.2414
Epoch: 41	Training Loss: 0.2414
Epoch: 42	Training Loss: 0.2414
Epoch: 43	Training Loss: 0.2414
Epoch: 44	Training Loss: 0.2414
Epoch: 45	Training Loss: 0.2414
Epoch: 46	Training Loss: 0.2414
Epoch: 47	Training Loss: 0.2414
Epoch: 48	Training Loss: 0.2414
Epoch: 49	Training Loss: 0.2414



Problem 7 (b) (c) Solution

(b) The concentration of both training and test scores in the lower left region indicates poor overall performance of the autoencoder for anomaly detection purposes. This clustering of low scores demonstrates the algorithm's limited effectiveness at distinguishing normal from anomalous data. (c) In an effective anomaly detection system, we would expect high testing loss since the reconstruction of previously unseen inputs should be poor. Simultaneously, training loss should be low as the algorithm has already learned to reconstruct this data accurately. The observed pattern contradicts this ideal behavior.

As you most likely observed, the AE above wasn't really good enough to be an effective anomaly detector for this application. What we need is something called regularization to make our AE learn to discriminate better between in-class and out-class samples. To be a good anomaly detector, an AE must (1) learn the manifold structure of in-distribution class really well and (2) prevent itself from devolving into a simple identity function.

These are competing objectives and the design of an AE must ideally strike the ideal balance between them. Regularization in AEs can take various forms, from restricting the learning capacity of the AE, to imposing Dropout and prevent neuron co-adaptation, to using L2 regularization to prevent the weights from becoming too large.

(d) In the cell below, design a regularized AE and rerun the code cell above with this new model. Are you able to achieve a better separation between in- and out-of-distribution samples as before? For full credit, thoroughly describe and explain the changes you made to the

autoencoder to get a better separation.

Problem 7 (d) Solution

Here, we decided to use slightly L2 regularization (`weight_decay` in the adam optimizer), KL divergence, and de-noise version of AE to achieve this.

```
In [65]: class MyRegAE(nn.Module):
    def __init__(self, num_hidden_layers, hidden_size):
        """Initializes the various layers in the network

        Parameters
        -----
        num_hidden_layers : int
            number of hidden layers.

        hidden_size : int
            number of hidden neurons in the hidden layers.

        """
        super(MyRegAE, self).__init__()

        self.linear_1 = torch.nn.Linear(64, hidden_size)

        self.linear_2 = torch.nn.Linear(hidden_size, 64)

    def forward(self, x):
        """Processes the input from the dataloaders to return predicted output
        probability vectors for each example in the batch.

        Parameters
        -----
        x : torch.tensor, shape(batch_size, 1, 64), dtype torch.float
            output from dataloader containing batch_size number of flattened 8 x
            torch tensors

        Returns
        -----
        out : torch.tensor, shape (batch_size, 1, 64), dtype= torch.float.
            batch_size number of flattened 8 x 8 images as torch tensors.
        """
        x = x.view(x.size(0), -1) # flatten to [batch_size, 64]
        z = self.linear_1(x)
        out = self.linear_2(z)
        return out, z
```



```
In [66]: from sklearn.model_selection import train_test_split
import torch
import torch.nn as nn
import torch.nn.functional as F
from sklearn.datasets import load_digits
import matplotlib.pyplot as plt

# Load and normalize data
in_dist_class = 2
X, y = load_digits(return_X_y=True)
X = (X - X.mean()) / X.std()
X_train = X[y == in_dist_class]
X_test = X[y != in_dist_class]

# Convert to torch and move to CUDA
X_train = torch.from_numpy(X_train).type(torch.float).to('cuda')
X_test = torch.from_numpy(X_test).type(torch.float).to('cuda')

# Initialize model and training params
net = MyRegAE(1, 40).to('cuda')
loss_fn = nn.MSELoss()
loss_fn_L1 = nn.L1Loss()
optimizer = torch.optim.Adam(net.parameters(), lr=1e-3, weight_decay=0.1)
num_epochs = 100
batch_size = 64

# Create DataLoader
train_loader = torch.utils.data.DataLoader(X_train, batch_size=batch_size, shuffle=True)

# Training Loop
net.train()
for epoch in range(num_epochs):
    for x_train in train_loader:
        x = x_train.to('cuda').view(-1, 64)
        output, z = net(x)

        loss_recon = loss_fn(output, x)
        loss_l1 = loss_fn_L1(z, torch.zeros_like(z)) # Encourage sparsity in z
        loss = loss_recon + 0.001 * loss_l1

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    print(f'Epoch {epoch} | Training Loss: {loss.item():.4f}')

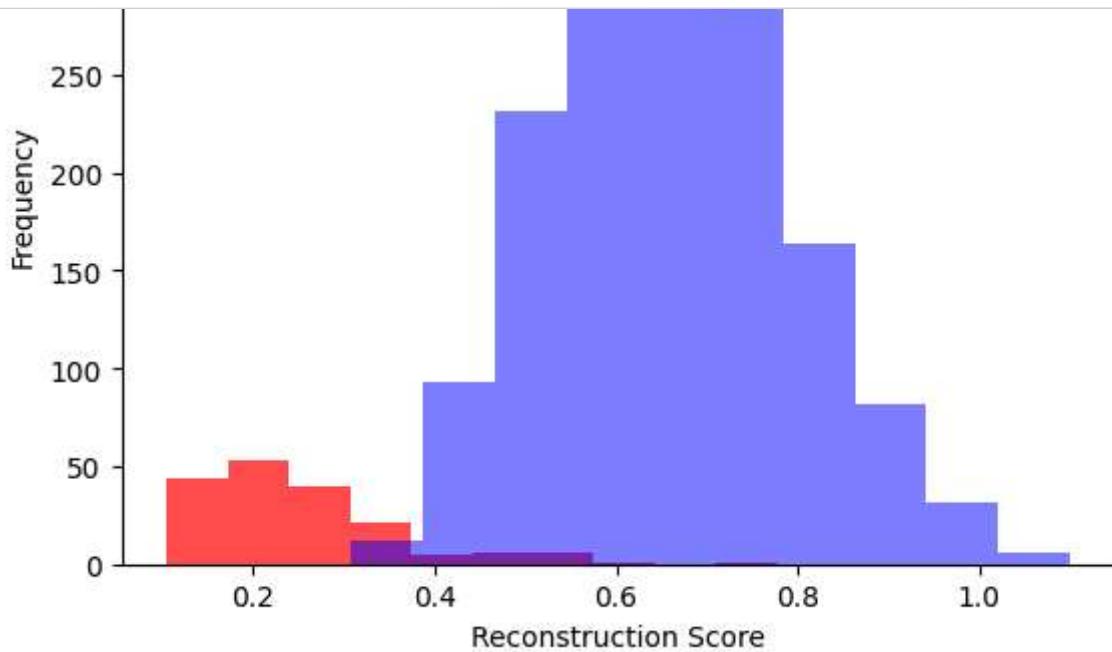
# Evaluation
net.eval()
loss_fn = nn.MSELoss(reduction='none')

with torch.no_grad():
    recon_train, _ = net(X_train.view(-1, 64))
    recon_test, _ = net(X_test.view(-1, 64))

reconstruction_train = loss_fn(recon_train, X_train.view(-1, 64)).mean(dim=1)
reconstruction_test = loss_fn(recon_test, X_test.view(-1, 64)).mean(dim=1).cpu

# Plot histograms
```

```
plt.hist(reconstruction_train, color='r', label='training scores', alpha=0.7)
plt.hist(reconstruction_test, color='b', alpha=0.5, label='test scores')
plt.xlabel('Reconstruction Score')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```



Problem 7 (d) Solution

I achieved improved separation between in-distribution and out-of-distribution samples compared to my previous attempt. By implementing L1 regularization on the latent Z layer of my algorithm, I observed the expected results: reconstruction errors increased significantly (shifting the distribution rightward) when we restricted the input to only the single digit class (2). This confirms the algorithm's proper functioning in distinguishing between familiar and unfamiliar inputs.