Assignment 5

● Graded

Student

Manish Kumar Krishne Gowda

Total Points

100 / 100 pts

Question 1

Exercise 1 — **40** / 40 pts

1.1 Task 1 — **10** / 10 pts

✔ **– 0 pts** Correct

– 10 pts No output/Incorrect

– 10 pts wrong pages selected

1.2 Task 2 — **20** / 20 pts

✔ **– 0 pts** Correct

– 10 pts No output/Incorrect

– 5 pts Loss is between 20-30

– 5 pts Noisy images are incorrect (Dataloader issue)

– 5 pts Reconstructed images are noisy

– 10 pts wrong pages selected

1.3 Task 3 — **10** / 10 pts

✔ **– 0 pts** Correct

– 15 pts Incorrect

Question 2

Exercise 2 — **50** / 50 pts

2.1 Task 1 — **15** / 15 pts

- ✔ **– 0 pts** Correct
- **– 10 pts** BCE incorrect
- **– 10 pts** KLD incorrect
- **– 15 pts** Wrong/not attempted
- **– 5 pts** Shape of merged lists and labels is wrong.

2.2 Task 2 — **35** / 35 pts

- **– 5 pts** scores incorrect
- ✔ **– 0 pts** Correct
- **– 10 pts** Reparametrization is wrong
- **– 10 pts** Encoder is incorrect
- **– 10 pts** Decoder is not correct
- **– 10 pts** Forward function is not correct
- **– 35 pts** Wrong/not implemented

Question 3

Exercise 3 — **10** / 10 pts

3.1 Task 1 — **10** / 10 pts

- ✔ **– 0 pts** Correct
- **– 5 pts** Clusters are not well separated
- **– 10 pts** Empty/wrong/not attempted
- **– 10 pts** Wrong/No Page Selected
- **– 10 pts** Mistake in Z or train selection

Question 4

Admin only (for late penalty) - assign to the first page — **0** / 0 pts

- ✔ **– 0 pts** Correct
- **– 10 pts** One day late
- **– 20 pts** Two days late
- **– 0 pts** Override late penalty
- **– 100 pts** More than two days late

No questions assigned to the following page.

ECE 57000 Assignment 5 Exercise

Your Name: Manish Kumar Krishne Gowda

Exercise 1: Define classifier that extracts latent representations and visualize representations (**40 points**)

The latent (i.e., hidden) representations generated by a deep neural network are very important concept in deep learning since the latent space is where the most significant features of the dataset are learned and extracted. In this homework, we will explore the latent representations of a classifier using clustering and nearest neighbor methods.

We provide the code for a simple residual CNN with batchnorm and data loaders.

- Here, we define a neural network block architecture that does batch normalization after each convolution layer and has a skip connection. You can read more about batch normalization and skip connections in these papers https://arxiv.org/pdf/1502.03167.pdf and https://arxiv.org/pdf/1512.03385.pdf, respectively.
- In this neural network, the block networks are designed so that the input dimension and the output dimension stay the same. This may not be an optimal design, but it will help us visualize the latent representation later.

In [1]:
```
import torch
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(device)
```

```
cuda
```

In [2]:
```
import torchvision
import torchvision.transforms as transforms
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import numpy as np

seed = 0
torch.manual_seed(seed)
torch.cuda.manual_seed(seed)
np.random.seed(seed)

class SimpleResidualBlock(nn.Module):
    def __init__(self, ch_in, mult=4):
        super().__init__()
```

No questions assigned to the following page.

```
        self.conv1 = nn.Conv2d(ch_in, mult * ch_in, kernel_size=3, stride=1, paddin
        self.bn1 = nn.BatchNorm2d(mult * ch_in)

        self.conv2 = nn.Conv2d(mult * ch_in, mult * ch_in, kernel_size=3, stride=1,
        self.bn2 = nn.BatchNorm2d(mult * ch_in)

        self.conv3 = nn.Conv2d(mult * ch_in, ch_in, kernel_size=3, stride=1, paddin
        self.bn3 = nn.BatchNorm2d(ch_in)

    def forward(self, x):
        x_ = x.clone()
        x_ = torch.relu(self.bn1(self.conv1(x_)))
        x_ = torch.relu(self.bn2(self.conv2(x_)))
        x_ = torch.relu(self.bn3(self.conv3(x_)))
        x = x + x_
        return x

class SimpleResNet(nn.Module):
    def __init__(self, ch_in, n_blocks=3):
        super().__init__()
        self.residual_layers = nn.ModuleList([SimpleResidualBlock(ch_in) for i in r
        self.maxpool = nn.MaxPool2d((2, 2))
        self.fc = nn.Linear(9, 10)

    def forward(self, x):
        for residual in self.residual_layers:
          x = residual(x)
          x = self.maxpool(x)
        x = x.view(x.shape[0], -1)  # Unravel tensor dimensions
        out = self.fc(x)
        return out
```

In [3]:

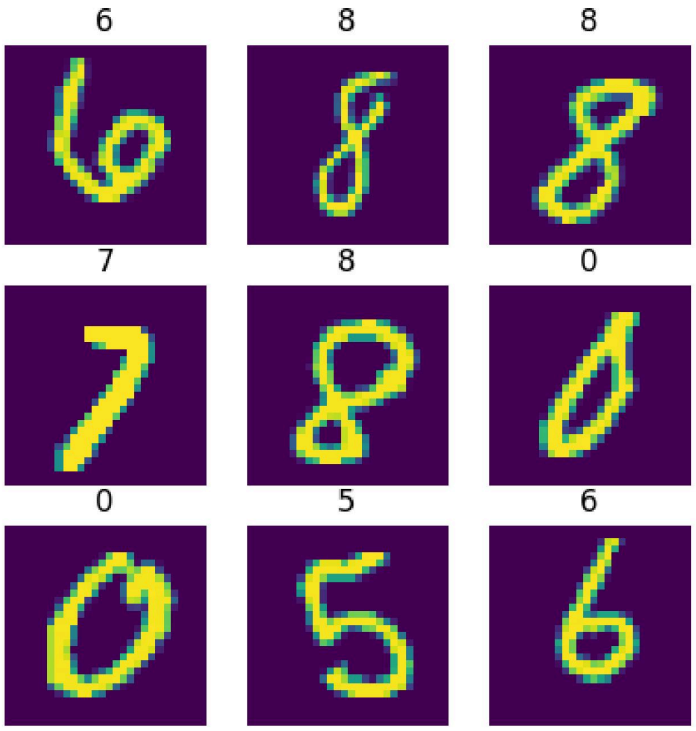
```
import torchvision.transforms as transforms
import matplotlib.pyplot as plt

# Create MNIST datasets
classes = np.arange(10)
transform = torchvision.transforms.Compose(
  [torchvision.transforms.ToTensor(),
   torchvision.transforms.Normalize((0.1307,),(0.3081,))])
train_dataset = torchvision.datasets.MNIST('./data', train=True, download=True, tra
test_dataset = torchvision.datasets.MNIST('./data', train=False, download=True, tra

# Create dataloaders
batch_size_train, batch_size_test = 64, 128
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size_tra
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size_test,

# Show sample images
batch_idx, (images, targets) = next(enumerate(train_loader))
fig, ax = plt.subplots(3,3,figsize = (5,5))
for i in range(3):
  for j in range(3):
    image = images[i*3+j].permute(1,2,0)
    image = image/2 + 0.5
    ax[i,j].imshow(image.squeeze(2))
```

No questions assigned to the following page.

```
    ax[i,j].set_title(f'{classes[targets[i*3+j]]}')
    ax[i,j].axis('off')
fig.show()
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Failed to download (trying next):
HTTP Error 403: Forbidden

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ubyte.gz
to ./data/MNIST/raw/train-images-idx3-ubyte.gz
100%|██████████| 9.91M/9.91M [00:01<00:00, 5.12MB/s]
Extracting ./data/MNIST/raw/train-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Failed to download (trying next):
HTTP Error 403: Forbidden

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ubyte.gz
to ./data/MNIST/raw/train-labels-idx1-ubyte.gz
100%|██████████| 28.9k/28.9k [00:00<00:00, 57.8kB/s]
Extracting ./data/MNIST/raw/train-labels-idx1-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Failed to download (trying next):
HTTP Error 403: Forbidden

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-ubyte.gz
to ./data/MNIST/raw/t10k-images-idx3-ubyte.gz
100%|██████████| 1.65M/1.65M [00:01<00:00, 1.28MB/s]
Extracting ./data/MNIST/raw/t10k-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Failed to download (trying next):
HTTP Error 403: Forbidden

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-labels-idx1-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-labels-idx1-ubyte.gz
to ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz
100%|██████████| 4.54k/4.54k [00:00<00:00, 3.04MB/s]
Extracting ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz to ./data/MNIST/raw
```

Question assigned to the following page: 1.1



Task 1: Inherit the original model class and define a new function returning the same output with the parent's forward function as well as intermediate representations (including the original input)

Specifically, the function below should return the original output from `forward` function of the parent's class and a list of intermediate representations `z_list`. `z_list` should be a Python list with 4 entries corresponding to the original batch and the batch **after each maxpool layer.**

Hints:

- Because of inheritance, you do not need to implement another `__init__` function. For those who are not familiar with inheritance, here is the link to get to know what Inheritance in Python is: https://www.geeksforgeeks.org/inheritance-in-python/.
- You will want to do the same computation as the original `forward` function but add some code to save intermediate representations, i.e., the code should output exactly the same thing as the original `forward` function but also return intermediate outputs.
- The output of this exercise should be:

```
Representation z0 batch shape = torch.Size([128, 1, 28, 28])
Representation z1 batch shape = torch.Size([128, 1, 14, 14])
Representation z2 batch shape = torch.Size([128, 1, 7, 7])
Representation z3 batch shape = torch.Size([128, 1, 3, 3])
```

Questions assigned to the following page: 1.1 and 1.2

- We provide a simple example for a linear model below.

In [ ]:
```python
# Trivial example below
class AffineModel(nn.Module):
    def __init__(self, A, b):
        super().__init__()
        self.A, self.b = A, b
    def forward(self, x):
        x = torch.matmul(x, self.A)
        return x + self.b
class ExtractAffineModel(AffineModel):
    def compute_and_extract_representations(self, x):
        z_list = [x]
        x = torch.matmul(x, self.A)
        z_list.append(x)
        return x + self.b, z_list
```

In [10]:
```python
class SimpleResNetWithRepresentations(SimpleResNet):
  def compute_and_extract_representations(self, x):
    # ----------- <Your code> ---------------
    z_list = [x]  # Store the original input as the first entry

    # Pass through each residual block and max pool
    for residual in self.residual_layers:
        x = residual(x)
        x = self.maxpool(x)
        z_list.append(x)  # Append each intermediate representation after max pooli

    # Final fully connected layer
    x = x.view(x.shape[0], -1)  # Flatten the output
    out = self.fc(x)
    # ----------- <End your code> ---------------
    return out, z_list

model = SimpleResNetWithRepresentations(ch_in=1)
model.to(device)
images, labels = next(iter(test_loader)) # get a batch
images = images.to(device)
# Check that outputs match
out, z_list = model.compute_and_extract_representations(images)
assert torch.all(model(images) == out), 'Outputs should be the same'
# Check shapes of representations
assert len(z_list) == 4, 'Should have length of 4'
assert torch.all(z_list[0] == images), 'First entry should be original data'
for zi, z in enumerate(z_list):
  print(f'Representation z{zi} batch shape = {z.shape}')
```

```
Representation z0 batch shape = torch.Size([128, 1, 28, 28])
Representation z1 batch shape = torch.Size([128, 1, 14, 14])
Representation z2 batch shape = torch.Size([128, 1, 7, 7])
Representation z3 batch shape = torch.Size([128, 1, 3, 3])
```

Task 2: Train the model

Question assigned to the following page: 1.2

Define the train function to train `model` for 4 epochs using the Adam optimizer with a learning rate of 0.01.

Define the test function to print average loss (use the variable `test_loss`) and accuracy (use the variable `correct`).

Test loss needs to be around 0.1-0.3 and Acccuracy needs to be higher than 90% after finishing training.

In [12]:
```
def train(epoch, model, optimizer):
    model.train() # we need to set the mode for our model
    for batch_idx, (images, targets) in enumerate(train_loader):
      # ----------- <Your code> ---------------
        images, targets = images.to(device), targets.to(device)  # Move data to the

        # Zero the parameter gradients
        optimizer.zero_grad()

        # Forward pass
        outputs = model(images)

        # Compute loss
        loss = F.cross_entropy(outputs, targets)

        # Backward pass and optimization
        loss.backward()
        optimizer.step()
    # ----------- <End Your code> ---------------
        if batch_idx % 100 == 0: # We visulize our output every 10 batches
            print(f'Epoch {epoch}: [{batch_idx*len(images)}/{len(train_loader.datas

def test(epoch, model):
  model.eval() # we need to set the mode for our model
  test_loss = 0 # sum up the loss value
  correct = 0 # sum up the corrected samples
  with torch.no_grad():
    for images, targets in test_loader:
      # ----------- <Your code> ---------------
        images, targets = images.to(device), targets.to(device)  # Move data to the

        # Forward pass
        outputs = model(images)

        # Compute and accumulate the loss
        test_loss += F.cross_entropy(outputs, targets, reduction='sum').item()

        # Compute the number of correct predictions
        _, predicted = torch.max(outputs, 1)
        correct += predicted.eq(targets).sum().item()
      # ----------- <End Your code> ---------------
  test_loss /= len(test_loader.dataset)
  print(f'Test result on epoch {epoch}: Avg loss is {test_loss}, Accuracy: {100.*co

import torch.optim as optim
```

Questions assigned to the following page: 1.2 and 1.3

```
optimizer = optim.Adam(model.parameters(), lr=0.01)

max_epoch = 4
for epoch in range(1, max_epoch+1):
  train(epoch, model, optimizer)
  test(epoch, model)
```

```
Epoch 1: [0/60000] Loss: 3.2035579681396484
Epoch 1: [6400/60000] Loss: 1.2096903324127197
Epoch 1: [12800/60000] Loss: 0.7747546434402466
Epoch 1: [19200/60000] Loss: 0.6352867484092712
Epoch 1: [25600/60000] Loss: 0.4514421820640564
Epoch 1: [32000/60000] Loss: 0.306108683347702
Epoch 1: [38400/60000] Loss: 0.44240206480026245
Epoch 1: [44800/60000] Loss: 0.36302992701530457
Epoch 1: [51200/60000] Loss: 0.23379819095134735
Epoch 1: [57600/60000] Loss: 0.33376815915107727
Test result on epoch 1: Avg loss is 0.2797416025161743, Accuracy: 91.85%
Epoch 2: [0/60000] Loss: 0.14114587008953094
Epoch 2: [6400/60000] Loss: 0.3022143542766571
Epoch 2: [12800/60000] Loss: 0.40931427478790283
Epoch 2: [19200/60000] Loss: 0.2100074291229248
Epoch 2: [25600/60000] Loss: 0.3176009953022003
Epoch 2: [32000/60000] Loss: 0.35170140862464905
Epoch 2: [38400/60000] Loss: 0.24400286376476288
Epoch 2: [44800/60000] Loss: 0.20154869556427002
Epoch 2: [51200/60000] Loss: 0.29714182019233704
Epoch 2: [57600/60000] Loss: 0.33015546202659607
Test result on epoch 2: Avg loss is 0.229794762301445, Accuracy: 92.62%
Epoch 3: [0/60000] Loss: 0.24645869433879852
Epoch 3: [6400/60000] Loss: 0.1858794391155243
Epoch 3: [12800/60000] Loss: 0.14809650182724
Epoch 3: [19200/60000] Loss: 0.3875899016857147
Epoch 3: [25600/60000] Loss: 0.16764521598815918
Epoch 3: [32000/60000] Loss: 0.3709743320941925
Epoch 3: [38400/60000] Loss: 0.380022794008255
Epoch 3: [44800/60000] Loss: 0.10537884384393692
Epoch 3: [51200/60000] Loss: 0.11486268043518066
Epoch 3: [57600/60000] Loss: 0.26833784580230713
Test result on epoch 3: Avg loss is 0.22193950469493867, Accuracy: 93.56%
Epoch 4: [0/60000] Loss: 0.21341831982135773
Epoch 4: [6400/60000] Loss: 0.21835964918136597
Epoch 4: [12800/60000] Loss: 0.2176124006509781
Epoch 4: [19200/60000] Loss: 0.08715561777353287
Epoch 4: [25600/60000] Loss: 0.1584017276763916
Epoch 4: [32000/60000] Loss: 0.1825229525566101
Epoch 4: [38400/60000] Loss: 0.2418297827243805
Epoch 4: [44800/60000] Loss: 0.19475096464157104
Epoch 4: [51200/60000] Loss: 0.19274525344371796
Epoch 4: [57600/60000] Loss: 0.2611961364746094
Test result on epoch 4: Avg loss is 0.16978728878498078, Accuracy: 94.76%
```

Task 3: Visualize the intermediate latent representations

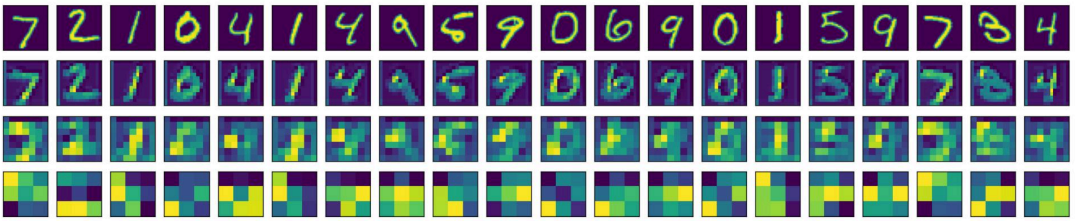
Question assigned to the following page: 1.3

- Plot the representations of 20 images from the test dataset in a supblots grid of shape (20, 4) (code already given for setting up these subplots) where the rows correspond to samples in the dataset and columns correspond to the representations produced by `compute_and_extract_representations`

Notes:

- We give code below for normalizing the image and plotting on an axis with a title.
- Make sure to set `model.eval()` when computing because of the batchnorm layers
- No title or ylabel is needed in this case.
- `z_list` is a list of 4 tensors of shape ([B, 1, 28, 28]). **Figure out how to pass through the assertion error and think why the batch dimension cannot be processed together.**

In [16]:
```
def plot_representation(z, ax):
  # Normalize image for visualization
  assert z.ndim == 3, 'Should be 3 dimensional tensor with C x H x W'
  z = (z - z.min())/(z.max() - z.min())
  if torch.is_tensor(z):  # Convert torch tensor to numpy if needed
    z = z.detach().cpu().numpy()
  ax.imshow(z.transpose((1,2,0)).squeeze(2))
  # Remove ticks and ticklabels to make plot clean
  ax.set_xticks([])
  ax.set_yticks([])
  ax.set_xticklabels([])
  ax.set_yticklabels([])

n_show = 20
fig, axes_mat = plt.subplots(4, n_show, figsize=[n_show, 4])
# ----------- <Your code> ---------------
model.eval()  # Set the model to evaluation mode

# Select 20 images from the test dataset
images, _ = next(iter(test_loader))  # Get a batch of images and labels
images = images[:n_show].to(device)  # Select the first 20 images

# Compute representations for the selected images
_, z_list = model.compute_and_extract_representations(images)

# Plotting in a (20, 4) grid: 20 images with 4 representations each
for i in range(n_show):
    for j in range(4):
        plot_representation(z_list[j][i], axes_mat[j, i])  # Access axes as [j, i]
# ----------- <End your code> ---------------
```



Question assigned to the following page: 2.1

Notice how the representations become more and more abstract as the depth increases.

Exercise 2: Clustering with different representations (50 points)

Task 1: Create simple numpy arrays of the representations

To perform further manipulations in numpy and scikit-learn, we will need to create simple numpy arrays for each representation. We provide the code for merging multiple batches. You will need to provide the code for extracting from the given data loader.

- Loop through the data loader and extract representations for each batch
- Append the labels and z_list to corresponding lists
- Break out of loop when the number extracted is n_extract or greater

The output of the merged lists should print the following for both train and test:

```
Types of merged lists
    [<class 'numpy.ndarray'>, <class 'numpy.ndarray'>, <class
'numpy.ndarray'>, <class 'numpy.ndarray'>]
Shapes of merged lists
    [(200, 1, 28, 28), (200, 1, 14, 14), (200, 1, 7, 7), (200, 1,
3, 3)]
Shape of merged labels
    (200,)
```

In [17]:
```python
def extract_numpy_representations(model, data_loader, n_extract):
  extracted_z_lists = []
  labels_list = []
  # ----------- <Your code> ---------------
  count = 0

  model.eval()  # Ensure model is in evaluation mode for consistent results

  with torch.no_grad():  # Disable gradient calculations
        for images, labels in data_loader:
            images = images.to(device)
            labels = labels.to(device)

            # Compute representations
            _, z_list = model.compute_and_extract_representations(images)

            # Append representations and labels to lists
            extracted_z_lists.append([z.detach() for z in z_list])
            labels_list.append(labels.detach())

            # Update count and check if we've reached the extraction limit
            count += images.size(0)
            if count >= n_extract:
```

Question assigned to the following page: 2.1

```
                break
    # ----------- <End your code> ---------------
    # Check extracted_z_lists (type should be tensor)
    print(f'Types of first batch\n    {[type(z) for z in extracted_z_lists[0]]}')
    print(f'Shapes of first batch\n    {[z.shape for z in extracted_z_lists[0]]}')

    # Merge extracted z_lists and labels and make numpy arrays
    z_list_merge_np = [
      np.vstack([
        z_list[i].detach().cpu().numpy()
        for z_list in extracted_z_lists
      ])[:n_extract]  # Extract up to n_extract
      for i in range(len(extracted_z_lists[0]))
    ]
    print(f'Types of merged lists\n    {[type(z) for z in z_list_merge_np]}')
    print(f'Shapes of merged lists\n    {[z.shape for z in z_list_merge_np]}')
    labels_merged_np = np.concatenate([
      labels.detach().cpu().numpy()
      for labels in labels_list
    ])[:n_extract]  # Extract up to n_extract
    print(f'Shape of merged labels\n    {labels_merged_np.shape}')
    return z_list_merge_np, labels_merged_np

# Extract train and test samples
z_list_train, labels_train = extract_numpy_representations(model, train_loader, n_e
z_list_test, labels_test = extract_numpy_representations(model, test_loader, n_extr
# Extract regular train and test
x_test = z_list_test[0]
x_train = z_list_train[0]
```

```
Types of first batch
    [<class 'torch.Tensor'>, <class 'torch.Tensor'>, <class 'torch.Tensor'>, <class
'torch.Tensor'>]
Shapes of first batch
    [torch.Size([64, 1, 28, 28]), torch.Size([64, 1, 14, 14]), torch.Size([64, 1, 7,
7]), torch.Size([64, 1, 3, 3])]
Types of merged lists
    [<class 'numpy.ndarray'>, <class 'numpy.ndarray'>, <class 'numpy.ndarray'>, <cla
ss 'numpy.ndarray'>]
Shapes of merged lists
    [(200, 1, 28, 28), (200, 1, 14, 14), (200, 1, 7, 7), (200, 1, 3, 3)]
Shape of merged labels
    (200,)
Types of first batch
    [<class 'torch.Tensor'>, <class 'torch.Tensor'>, <class 'torch.Tensor'>, <class
'torch.Tensor'>]
Shapes of first batch
    [torch.Size([128, 1, 28, 28]), torch.Size([128, 1, 14, 14]), torch.Size([128, 1,
7, 7]), torch.Size([128, 1, 3, 3])]
Types of merged lists
    [<class 'numpy.ndarray'>, <class 'numpy.ndarray'>, <class 'numpy.ndarray'>, <cla
ss 'numpy.ndarray'>]
Shapes of merged lists
    [(200, 1, 28, 28), (200, 1, 14, 14), (200, 1, 7, 7), (200, 1, 3, 3)]
Shape of merged labels
    (200,)
```

Question assigned to the following page: 2.2

Task 2: Perform K-means clustering on different representations

In this task, we will perform kmeans clustering on each of the latent representations of the test set and then evaluate the clustering based on the true class labels. A good discussion of clustering metrics can be found in scikit-learn's documentation on clustering metrics.

- Using scikit-learn's `sklearn.cluster.KMeans` estimator, perform kmeans with $k = 10$ and `random_state=0` on the latent representations and extract the cluster labels.
- Use `sklearn.metrics.adjusted_rand_score` to compute a score to evaluate the clustering based on the true class labels.

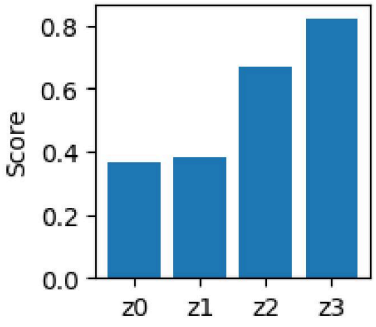
Notes:

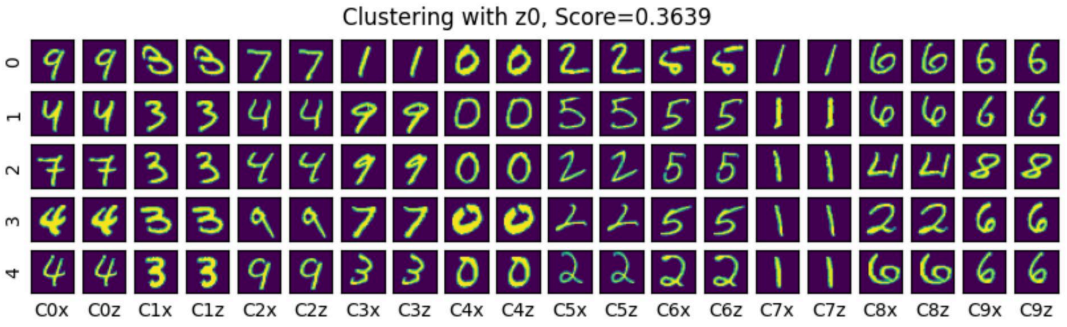
- You will need to reshape the tensors into matrices immediately before passing into sklearn functions (you should keep the original data as is so that the images can be plotted, but just reshape immediately before passing into scikit-learn functions). Specifically, the arrays will have shape (B, C, H, W) and you should reshape to (B, C*H*W) before passing to scikit-learn functions.
- We provide code for plotting and evaluating your clustering.
- Note that clustering is unsupervised. What we're plotting here is the ten different clusters, not the ten different categories of true labels. Thus, the cluster index in the plotted image is not necessarily matched to the true label.
- Sometimes the `plot_cluster` will have white boxes if there are less than 5 samples in that cluster. Generally, if you use `n_clusters=10` for the clustering tasks, you will have none or only a few white boxes, which is okay.

In [19]:
```
def plot_cluster(cluster_labels, z_test, title):
  # Plot the top images in each cluster both in original space and latent represent
  n_samples_show, n_clusters = 5, 10
  nr, nc = n_samples_show, 2*n_clusters
  fig, axes_mat = plt.subplots(nr, nc, figsize=np.array([nc, nr])/2)
  axes_mat_list = np.split(axes_mat, n_clusters, axis=1)
  for ci, axes_mat in enumerate(axes_mat_list): # Loop over clusters
    sel = cluster_labels==ci
    z_cluster = z_test[sel][:n_samples_show]
    x_cluster = x_test[sel][:n_samples_show]
    for test_i, (z, x, axes) in enumerate(zip(z_cluster, x_cluster, axes_mat)):
      plot_representation(x, axes[0])
      plot_representation(z, axes[1])
      if ci == 0:
        axes[0].set_ylabel(test_i)
      if test_i == len(axes_mat)-1:
        axes[0].set_xlabel(f'C{ci}x')
        axes[1].set_xlabel(f'C{ci}z')
  fig.suptitle(title)
  plt.show()
```

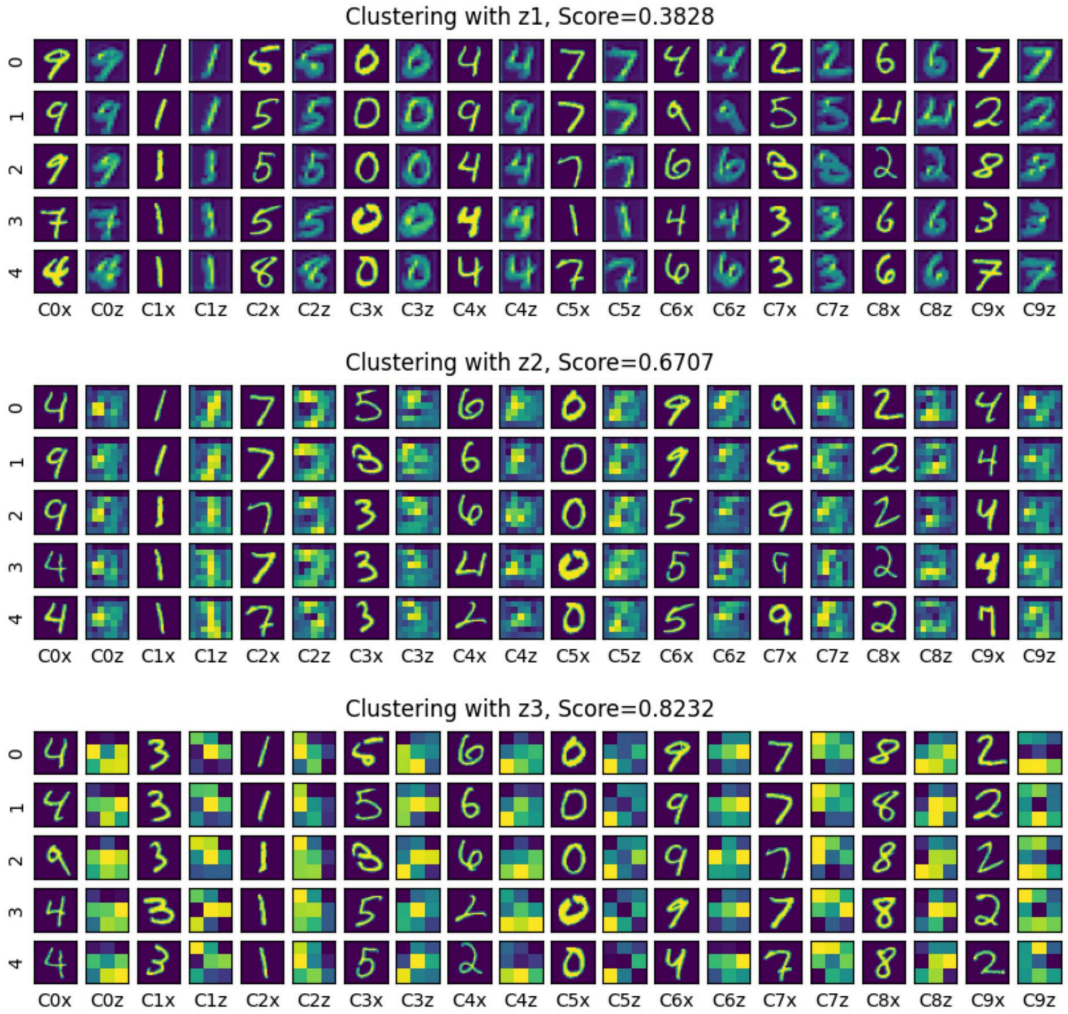
Question assigned to the following page: 2.2

In [20]:
```python
from sklearn.cluster import KMeans
from sklearn.metrics import adjusted_rand_score

def cluster_and_score(z, true_labels):
  # ----------- <Your code> ---------------
  # Reshape (B, C, H, W) -> (B, C*H*W) for clustering
  B, C, H, W = z.shape
  z_reshaped = z.reshape(B, C * H * W)

  # Perform k-means clustering with 10 clusters (assuming 10 classes)
  kmeans = KMeans(n_clusters=10, random_state=0)
  cluster_labels = kmeans.fit_predict(z_reshaped)

  # Calculate clustering score using Adjusted Rand Index
  score = adjusted_rand_score(true_labels, cluster_labels)
  # ----------- <End your code> ---------------
  return cluster_labels, score

fig = plt.figure(figsize=(2,2))
plt.bar([f'z{zi}' for zi in range(len(z_list_test))],
        [cluster_and_score(z_test, labels_test)[1] for z_test in z_list_test])
plt.ylabel('Score')
for zi, z_test in enumerate(z_list_test):
  cluster_labels, score = cluster_and_score(z_test, labels_test)
  plot_cluster(cluster_labels, z_test, f'Clustering with z{zi}, Score={score:.4f}')
```





Questions assigned to the following page: 3.1 and 2.2



Notice how the 3x3 pattern for the last representation looks similar across the samples.

Exercise 3: Nearest neigbhors methods using representations (10 points)

Task 1: Compute and plot nearest neigbhors in different representations

We will now compute the 1 nearest neighbor (i.e., `n_neighbors=1`) of test points compared to train points in different representations.

- Loop through the representations for the train and test numpy arrays (i.e., `z_list_train` and `z_list_test`).
- For each representaiton from the different layers, compute the *training* indices corresponding to the nearest neighbor of first 15 *testing* indices.

Question assigned to the following page: 3.1

- Plot the neighbors by passing the test indices and corresponding nearest neighbor training indices along with the corresponding train and test representations and a title that describes which representation into `plot_neighbor`.

Notes:

- See note above about reshaping tensors immediately before passing to scikit-learn functions which expect a matrix.
- The `sklearn.neighbors.NearestNeighbors` class and the `kneighbors` method may be very helpful. The data that is passed to `fit` will be the training data and the data passed to `kneighbors` should be the new test data.
- The test indices should just be `np.arange(15)` assuming that you find the nearest training points for the first 15 points in the test dataset.

In [21]:
```
def plot_neigbhor(test_ind, nearest_train_ind, z_test, z_train, title):
  '''
  Plots the original test image, the test image representation,
    the nearest train image representation, the nearest original train image.
  '''
  assert len(test_ind) == len(nearest_train_ind), 'Test and train indices should be
  n_test = len(test_ind)
  fig, axes_mat = plt.subplots(4, n_test, figsize=np.array([n_test, 4])/2)
  for test_i, nearest_train_i, axes in zip(test_ind, nearest_train_ind, axes_mat.T)
    plot_representation(x_test[test_i], axes[0])
    plot_representation(z_test[test_i], axes[1])
    plot_representation(z_train[nearest_train_i], axes[2])
    plot_representation(x_train[nearest_train_i], axes[3])
    if test_i == 0:
      for lab, ax in zip(['X Tst', 'Z Tst', 'Z Tr', 'X Tr'], axes):
        ax.set_ylabel(lab)
  fig.suptitle(title)

from sklearn.neighbors import NearestNeighbors

# ----------- <Your code> ---------------
n_neighbors = 1
n_test_samples = 15  # Number of test samples to evaluate for nearest neighbors

# Loop over each representation in the train and test lists
for zi, (z_train, z_test) in enumerate(zip(z_list_train, z_list_test)):
    # Reshape representations to (B, C*H*W)
    B_train, C_train, H_train, W_train = z_train.shape
    B_test, C_test, H_test, W_test = z_test.shape
    z_train_reshaped = z_train.reshape(B_train, C_train * H_train * W_train)
    z_test_reshaped = z_test.reshape(B_test, C_test * H_test * W_test)

    # Initialize nearest neighbor model and fit on the training data
    nbrs = NearestNeighbors(n_neighbors=n_neighbors, algorithm='auto').fit(z_train_

    # Find nearest neighbors for the first 15 test samples
    test_indices = np.arange(n_test_samples)
    distances, nearest_train_indices = nbrs.kneighbors(z_test_reshaped[test_indices
```

Question assigned to the following page: 3.1

```
    nearest_train_indices = nearest_train_indices.flatten()  # Get nearest neighbor

    # Plot neighbors for each representation layer
    plot_neigbhor(test_indices, nearest_train_indices, z_test, z_train, title=f'Nea
# ----------- <End your code> ---------------
```