Enter your username (used for marking):

```
username = 'acq22xc'
```

COM4509/6509 Coursework Part 2

Hello, This is the *second* of the two parts. Each part accounts for 50% of the overall coursework mark and this part has a total of 50 marks available. Attempt as much of this as you can, each of the questions are self-contained and contain some easier and harder bits so even if you can't complete Q1 straight away then you may still be able to progress with the other questions.

Overview

This part of the assignment will cover:

- Logistic regression and PCA: 13 Marks, lecture 6 and 8.
- Neural networks: 16 Marks, lecture 7 and 8.
- Auto-encoders: 16 Marks, lecture 8 and 9.

What to submit

- You need to submit **two jupyter notebooks** (not zipped together) and a **pdf** copy of part 2, named:

```
assignment_part1_[username].ipynb
assignment_part2_[username].ipynb
assignment_part2_[username].pdf
```

replacing `[username]` with your username, e.g. `abc18de`.

- **Please execute the cells before your submission**. The **pdf** copy will be used as a backup in case the data gets corrupted and since we cannot run all the notebooks during marking. The best way to get a pdf is using Jupyter Notebook locally but if you are using Google Colab and are unable to download it to use Jupyter then you can use the Google Colab *file* → *print* to get a pdf copy.
- **Please do not upload** the data files used in this Notebook. We just want the two python notebooks *and the pdf*.

Assessment Criteria

- The marks are indicated for each part: You'll get marks for correct code that does what is asked and gets the right answer. **These contribute 45**. You should make sure any figures are plotted properly with axis labels and figure legends.
- There are also **5 marks for "Code quality"** (includes both readability and efficiency).

Late submissions

We follow the department's guidelines about late submissions, Undergraduate handbook link. PGT handbook link.

Use of unfair means

This is an individual assignment, while you may discuss this with your classmates, please make sure you submit your own code. You are allowed to use code from the labs as a basis of your submission.

"Any form of unfair means is treated as a serious academic offence and action may be taken under the Discipline Regulations." (from the students Handbook).

Reproducibility and readibility

Whenever there is randomness in the computation, you MUST set a random seed for reproducibility. Use your UCard number XXXXXXXXX (or the digits in your registration number if you do not have one) as the random seed throughout this assignment. You can set the seeds using torch.manual_seed(XXXXX) and np.random.seed(XXXXX). Answers for each question should be clearly indicated in your notebook. While code segments are indicated for answers, you may use more cells as necessary. All code should be clearly documented and explained. Note: You will make several design choices (e.g. hyperparameters) in this assignment. There are no “standard answers”. You are encouraged to explore several design choices to settle down with good/best ones, if time permits.

Question 1: Logistic regression and PCA [13 marks]

MedMNIST is a collection of healthcase based datasets that are pre-processed to match to format of the original MNIST dataset. In this questions, you will perform logistic regression and dimension reduction using PCA on the **PneumoniaMNIST** dataset from the MedMNIST. The task for this dataset is to detect whether a chest X-ray shows signs of Pneumonia or not and is therefore a binary classification task.

1.1: Data download [1 mark]

The code cell belows provides the code to download the dataset as a compressed numpy file directly from the MedMNIST website. If you prefer, you can follow the instructions at https://github.com/MedMNIST/MedMNIST to download and load the data.

```
import numpy as np
import urllib.request
import os
```

```
# Download the dataset to the local folder
urllib.request.urlretrieve('https://zenodo.org/record/6496656/files/pneumoniamnist.

# Load the compressed numpy array file
dataset = np.load('./pneumoniamnist.npz')

# The loaded dataset contains each array internally
for key in dataset.keys():
    print(key, dataset[key].shape, dataset[key].dtype)
```

```
train_images (4708, 28, 28) uint8
val_images (524, 28, 28) uint8
test_images (624, 28, 28) uint8
train_labels (4708, 1) uint8
val_labels (524, 1) uint8
test_labels (624, 1) uint8
```

1.1a After downloading the data, merge the validation set into the training set and reshape the images so that each is a 1D array. Then scale the pixel values so they are in the range [0,1].

```
# Write your code here.
# for key in dataset.keys():
#     print(key)
# print(dataset['train_images'])
train_images = dataset['train_images']
val_images = dataset['val_images']
test_images = dataset['test_images']
train_labels = dataset['train_labels']
val_labels = dataset['val_labels']
test_labels = dataset['test_labels']
train_images = np.concatenate((train_images,val_images),axis=0).reshape(5232,28*28)
train_labels = np.concatenate((train_labels,val_labels),axis=0)
test_images = test_images.reshape(624,28*28)
train_images = train_images/255
test_images = test_images/255
```

1.2: Dimensional reduction and training [6 marks]

1.2a Using the Scikit-learn PCA class, transform the training and test data into **at least seven** different sets of reduced dimensions, i.e create 7 alternate datsets with $(k_1, k_2, \ldots, k_7)$ number of features. **Briefly explain** your choice reduced features. Keep a copy of the unreduced data so that in total you have **eight** datasets.

You should fit the tranformation based on the training data and use that to transform the test data. You can find details of the PCA transformation class here.

```
# Write your code here.
from sklearn.decomposition import PCA
pca_2 = PCA(n_components=2)                              #The PCA algorithm is used to
train_images_2 = pca_2.fit_transform(train_images)
```

```
test_images_2 = pca_2.fit_transform(test_images)
pca_4 = PCA(n_components=4)
train_images_4 = pca_4.fit_transform(train_images)
test_images_4 = pca_4.fit_transform(test_images)
pca_8 = PCA(n_components=8)
train_images_8 = pca_8.fit_transform(train_images)
test_images_8 = pca_8.fit_transform(test_images)
pca_16 = PCA(n_components=16)
train_images_16 = pca_16.fit_transform(train_images)
test_images_16 = pca_16.fit_transform(test_images)
pca_32 = PCA(n_components=32)
train_images_32 = pca_32.fit_transform(train_images)
test_images_32 = pca_32.fit_transform(test_images)
pca_64 = PCA(n_components=64)
train_images_64 = pca_64.fit_transform(train_images)
test_images_64 = pca_64.fit_transform(test_images)
pca_128 = PCA(n_components=128)
train_images_128 = pca_128.fit_transform(train_images)
test_images_128 = pca_128.fit_transform(test_images)
print(train_images_2.shape)
print(train_images_4.shape)
print(train_images_8.shape)
print(train_images_16.shape)
print(train_images_32.shape)
print(train_images_64.shape)
print(train_images_128.shape)
```

```
(5232, 2)
(5232, 4)
(5232, 8)
(5232, 16)
(5232, 32)
(5232, 64)
(5232, 128)
```

1.2b Train **eight** logistic regression classifiers (LRC): one on the original features (unreduced), and seven on PCA features with seven different dimensions in 1.2a, i.e., LRC on k_1 PCA features; LRC on k_2 PCA features; ..., LRC on k_7 PCA features and LRC on the unreduced data. You will need to decide on any options for the logistic regression fitting and **explain** which choices you make. You can use the Scikit Learn Logistic Regression classifier, further information is given here.

```
# Write your code here.
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()   # Logistic Regression model
logreg.fit(train_images, train_labels)
logreg_2 = LogisticRegression()
logreg_2.fit(train_images_2, train_labels)
logreg_4 = LogisticRegression()
logreg_4.fit(train_images_4, train_labels)
logreg_8 = LogisticRegression()
logreg_8.fit(train_images_8, train_labels)
```

```
logreg_16 = LogisticRegression()
logreg_16.fit(train_images_16, train_labels)
logreg_32 = LogisticRegression()
logreg_32.fit(train_images_32, train_labels)
logreg_64 = LogisticRegression()
logreg_64.fit(train_images_64, train_labels)
logreg_128 = LogisticRegression()
logreg_128.fit(train_images_128, train_labels)
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/utils/validation.py:993: DataCc
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_logistic.py:814:
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.8/dist-packages/sklearn/utils/validation.py:993: DataCc
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.8/dist-packages/sklearn/utils/validation.py:993: DataCc
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.8/dist-packages/sklearn/utils/validation.py:993: DataCc
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.8/dist-packages/sklearn/utils/validation.py:993: DataCc
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.8/dist-packages/sklearn/utils/validation.py:993: DataCc
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.8/dist-packages/sklearn/utils/validation.py:993: DataCc
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.8/dist-packages/sklearn/utils/validation.py:993: DataCc
  y = column_or_1d(y, warn=True)
LogisticRegression()
```

1.3: Model evaluation [6 marks]

1.3b For each of the trained classifiers in 1.2b, calculate the classification accuracy on the training data and the test data. Extract the total explained variance by summing the `PCA.explained_variance_ratio_` for each of your PCA transformations. **Plot** the training accuracy and test accuracy against the total explained variance at each k_n. You should include the results for the case trained on the original features, which corresponds to a total explained variance of 1.

```
# Write your code here.
from sklearn.metrics import accuracy_score
y_train_pred = logreg.predict(train_images)
y_pred = logreg.predict(test_images)
acc_train = accuracy_score(train_labels, y_train_pred)
acc_test = accuracy_score(test_labels, y_pred)
print(acc_train)
print(acc_test)
```

```
y_train_pred_2 = logreg_2.predict(train_images_2)
y_pred_2 = logreg_2.predict(test_images_2)
acc_train_2 = accuracy_score(train_labels, y_train_pred_2)
acc_test_2 = accuracy_score(test_labels, y_pred_2)
print(acc_train_2)
print(acc_test_2)

y_train_pred_4 = logreg_4.predict(train_images_4)
y_pred_4 = logreg_4.predict(test_images_4)
acc_train_4 = accuracy_score(train_labels, y_train_pred_4)
acc_test_4 = accuracy_score(test_labels, y_pred_4)
print(acc_train_4)
print(acc_test_4)

y_train_pred_8 = logreg_8.predict(train_images_8)
y_pred_8 = logreg_8.predict(test_images_8)
acc_train_8 = accuracy_score(train_labels, y_train_pred_8)
acc_test_8 = accuracy_score(test_labels, y_pred_8)
print(acc_train_8)
print(acc_test_8)

y_train_pred_16 = logreg_16.predict(train_images_16)
y_pred_16 = logreg_16.predict(test_images_16)
acc_train_16 = accuracy_score(train_labels, y_train_pred_16)
acc_test_16 = accuracy_score(test_labels, y_pred_16)
print(acc_train_16)
print(acc_test_16)

y_train_pred_32 = logreg_32.predict(train_images_32)
y_pred_32 = logreg_32.predict(test_images_32)
acc_train_32 = accuracy_score(train_labels, y_train_pred_32)
acc_test_32 = accuracy_score(test_labels, y_pred_32)
print(acc_train_32)
print(acc_test_32)

y_train_pred_64 = logreg_64.predict(train_images_64)
y_pred_64 = logreg_64.predict(test_images_64)
acc_train_64 = accuracy_score(train_labels, y_train_pred_64)
acc_test_64 = accuracy_score(test_labels, y_pred_64)
print(acc_train_64)
print(acc_test_64)

y_train_pred_128 = logreg_128.predict(train_images_128)
y_pred_128 = logreg_128.predict(test_images_128)
acc_train_128 = accuracy_score(train_labels, y_train_pred_128)
acc_test_128 = accuracy_score(test_labels, y_pred_128)
print(acc_train_128)
print(acc_test_128)
```

```
0.9692278287461774
0.844551282051282
0.8797782874617737
0.3717948717948718
0.8920107033639144
0.375
```

```
0.9235474006116208
0.40224358974358976
0.9420871559633027
0.40224358974358976
0.9545107033639144
0.40384615384615385
0.9629204892966361
0.40064102564102566
0.9648318042813455
0.40384615384615385
```

```
print(sum(pca_2.explained_variance_ratio_))
print(sum(pca_4.explained_variance_ratio_))
print(sum(pca_8.explained_variance_ratio_))
print(sum(pca_16.explained_variance_ratio_))
print(sum(pca_32.explained_variance_ratio_))
print(sum(pca_64.explained_variance_ratio_))
print(sum(pca_128.explained_variance_ratio_))
```

```
0.4365232044059972
0.5935461134666948
0.7195590234447479
0.8297433237288396
0.9046881066248288
0.9500395272488441
0.9793094028018987
```

1.3b Describe at least **two** relevant observations from the evaluation results above.

```
# Write your answer here.
#As can be seen from our results above, when there are 64 principal components, the
#Accuracy in the training set is much higher than that in the test set
```

Question 2: Convolutional neural networks for image recognition [16 marks]

Fashion-MNIST is a dataset of Zalando's article images. It consists of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes: 0=T-shirt/top; 1=Trouser; 2=Pullover; 3=Dress; 4=Coat; 5=Sandal; 6=Shirt; 7=Sneaker; 8=Bag; 9=Ankle boot.

It is available online at https://github.com/zalandoresearch/fashion-mnist but here we will use the version built into PyTorch as part of the TorchVision library see here for documentation.

In this question, you should PyTorch to train various forms of neural network models to classify these images. You can refer to Lab 7 on how to define and train neural networks with PyTorch.

2.1: Data download and inspection [3 marks]

2.1a Use the PyTorch Torchvision API to load both the train and test parts of the Fashion-MNIST dataset. You can use the code used in Lab 7 to load the CIFAR10 as a basis for this.

```
# Write your code here.
%matplotlib inline
import torch
import torchvision
from torch.utils import data
from torchvision import transforms
trans = transforms.ToTensor()
mnist_train = torchvision.datasets.FashionMNIST(
    root=r"FashionMnist", train=True, transform=trans, download=True)
mnist_test = torchvision.datasets.FashionMNIST(
    root=r"FashionMnist", train=False, transform=trans, download=True)
```

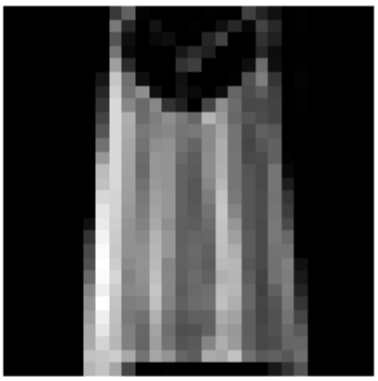
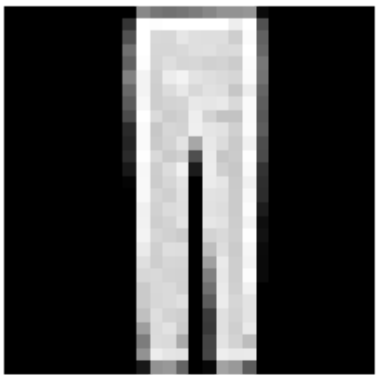
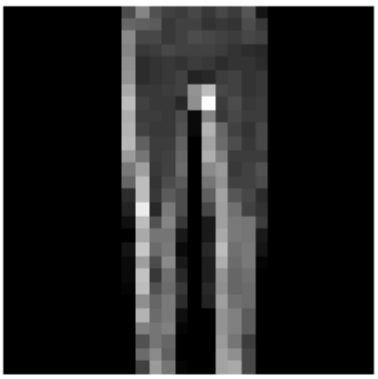
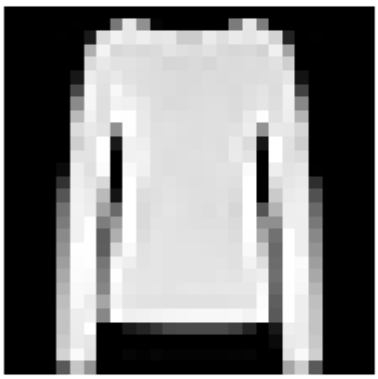
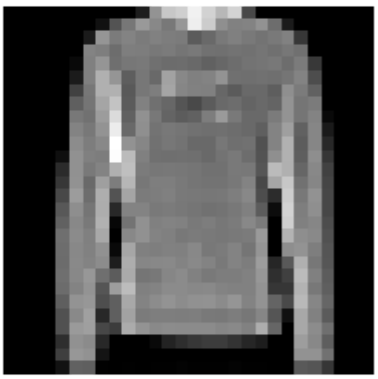
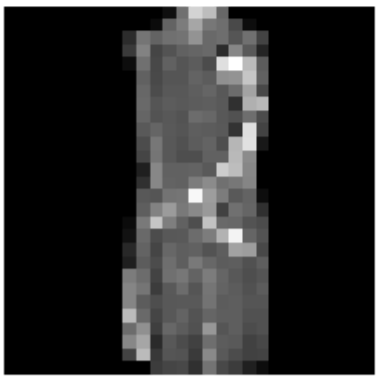
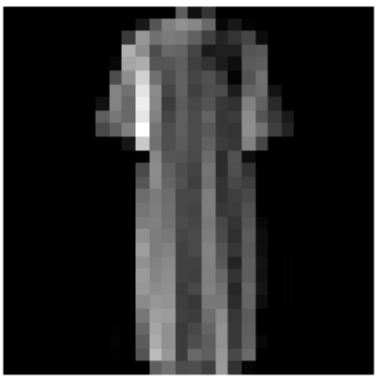
2.1b Use the `torch.utils.data.random_split` function to split the 60,000 training set into 2 subsets: the first part will be used for training, the second part will be used for validation. You must choose a sensible split of this into the training and validation sets. Create a DataLoader for each of the train, validation, and test splits.

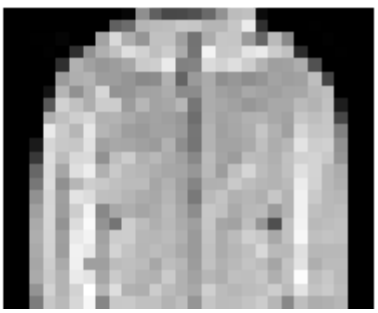
```
# Write your code here.
from torch.utils.data import random_split
import torch.utils.data as Data
train_dataset, val_dataset = random_split(
    dataset=mnist_train,
    lengths=[50000, 10000],
    generator=torch.Generator().manual_seed(0)
)
train_loader = Data.DataLoader(dataset=train_dataset,  # Imported data sets
                               batch_size=64,  # Number of samples contained in
                               shuffle=True,  # Reordering of data sets
                               num_workers=0,  # Loading number of processes op
                               )
val_loader = Data.DataLoader(dataset=train_dataset,
                               batch_size=1,
                               shuffle=True,
                               num_workers=0,
                               )
test_loader = Data.DataLoader(dataset=train_dataset,
                               batch_size=1,
                               shuffle=True,
                               num_workers=0,
                               )
```

2.1c Display 2 example images from each of the classes (20 images in total).

```
# Write your code here.
import matplotlib.pyplot as plt
```

```
r,c=10,2
w,h=28,28
fig,axs=plt.subplots(r,c,figsize=(10,40))
for i in range(10):
    k=0
    image,label = next(iter(train_loader))
    for num,j in enumerate(label):
        if j==i:
            axs[i,k].imshow(image[num][0],cmap='gray')
            axs[i,k].axis('off')
            k+=1
        if k==2:
            break
```



2.2: Network training [8 marks]

In this section you will train a set of neural network models to classify the Fashion-MNIST data set. Only the number of convolutional (Conv) layers and the number of fully connected (FC) layers will be specified below. You are free to design other aspects of the network. For example, you can use other types of operation (e.g. padding), layers (e.g. pooling, or preprocessing (e.g. augmentation), and you choose the number of units/neurons in each layer. Likewise, you may choose the number of epochs and many other settings according to your accessible computational power. You should choose sensible values for the batch size and learning rate. If you wish, you may use alternate optimisers, such as Adam.

When training each model you should keep track of the following values:

1. Training accuracy
2. Validation accuracy
3. Test accuracy

Remember the accuracy is the number of correct classifications out of that portion of the dataset.

2.2a Train a neural network composed of **2 fully connected layers** with an activation function of your choice. Train the model on the training set, use the validation set to choose the best design among **at least three different** choices, and test the chosen model on the test set.

Remember that your dataloader will give you a 2D image. The CNNs can process these but your fully connected (`nn.Linear`) layers are expecting each sample to be a vector.

```
# Write your code here.
```

```
import torch.nn as nn
import torch.optim as optim
class net(nn.Module):
    def __init__(self):
        super(net, self).__init__()
        self.fc = nn.Sequential(
            nn.Linear(28*28, 512),
            nn.ReLU(),
            nn.Linear(512, 10)
        )
    def forward(self, x):
        x = x.view(-1, 28 * 28)
        x = self.fc(x)
        return x
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = net().to(device)
criterion = torch.nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(),
                      lr=0.001,
                      momentum=0.9,
                      weight_decay=1e-5)
max_acc = 0
for epoch in range(10):
    for idx, (images, labels) in enumerate(train_loader):
        images, labels = images.to(device), labels.to(device)
        output = model(images)
        loss = criterion(output,labels)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    correct=0
    total=0
    with torch.no_grad():
        for data in val_loader:
            inputs,target=data
            inputs,target = inputs.to(device), target.to(device)
            outputs=model(inputs)
            _,predicted=torch.max(outputs.data,dim=1)
            total+=target.size(0)
            correct+=(predicted==target).sum().item()
    acc = 100*correct/total
    print('Accuracy on val set:{}'.format(acc))
    if acc>max_acc:
        max_acc = acc
        torch.save(model.state_dict(),"best_model.pth" )
```

```
Accuracy on val set:73.874
Accuracy on val set:79.658
Accuracy on val set:81.514
Accuracy on val set:82.632
Accuracy on val set:83.36
Accuracy on val set:83.814
Accuracy on val set:84.4
Accuracy on val set:84.428
```

```
Accuracy on val set:84.846
Accuracy on val set:84.958
```

```
state_dict = torch.load("best_model.pth")
model.load_state_dict(state_dict, strict=False)
for data in train_loader:
    inputs,target=data
    inputs,target = inputs.to(device), target.to(device)
    outputs=model(inputs)
    _,predicted=torch.max(outputs.data,dim=1)
    total+=target.size(0)
    correct+=(predicted==target).sum().item()
train_acc = 100*correct/total
print('Accuracy on train set:{}'.format(train_acc))
for data in val_loader:
    inputs,target=data
    inputs,target = inputs.to(device), target.to(device)
    outputs=model(inputs)
    _,predicted=torch.max(outputs.data,dim=1)
    total+=target.size(0)
    correct+=(predicted==target).sum().item()
val_acc = 100*correct/total
print('Accuracy on val set:{}'.format(val_acc))
for data in test_loader:
    inputs,target=data
    inputs,target = inputs.to(device), target.to(device)
    outputs=model(inputs)
    _,predicted=torch.max(outputs.data,dim=1)
    total+=target.size(0)
    correct+=(predicted==target).sum().item()
test_acc = 100*correct/total
print('Accuracy on test set:{}'.format(test_acc))
```

```
Accuracy on train set:84.958
Accuracy on val set:84.958
Accuracy on test set:84.958
```

2.2b Define and train using a neural network composed of **2 convolutional layers and 2 fully connected layers**. Train the model on the training set, use the validation set to choose the best design among **at least three different** choices, and test the chosen model on the test set.

```
# Write your code here.
class net(nn.Module):
    def __init__(self):
        super(net, self).__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(1, 32, 5),
            nn.ReLU(),
            nn.MaxPool2d(2, stride = 2),
            nn.Dropout(0.3),
            nn.Conv2d(32, 64, 5),
            nn.ReLU(),
```

```
            nn.MaxPool2d(2, stride = 2),
            nn.Dropout(0.3)
        )
        self.fc = nn.Sequential(
            nn.Linear(64 * 4 * 4, 512),
            nn.ReLU(),
            nn.Linear(512, 10)
        )
    def forward(self, x):
        x = self.conv(x)
        x = x.view(-1, 64 * 4 * 4)
        x = self.fc(x)
        return x

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = net().to(device)
criterion = torch.nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(),
                      lr=0.001,
                      momentum=0.9,
                      weight_decay=1e-5)
max_acc = 0
for epoch in range(10):
    for idx, (images, labels) in enumerate(train_loader):
        images, labels = images.to(device), labels.to(device)
        output = model(images)
        loss = criterion(output,labels)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    correct=0
    total=0
    with torch.no_grad():
        for data in val_loader:
            inputs,target=data
            inputs,target = inputs.to(device), target.to(device)
            outputs=model(inputs)
            _,predicted=torch.max(outputs.data,dim=1)
            total+=target.size(0)
            correct+=(predicted==target).sum().item()
    acc = 100*correct/total
    print('Accuracy on val set:{}'.format(acc))
    if acc>max_acc:
        max_acc = acc
        torch.save(model.state_dict(),"best_model.pth" )
```

```
Accuracy on val set:66.566
Accuracy on val set:72.966
Accuracy on val set:74.672
Accuracy on val set:77.074
Accuracy on val set:78.198
Accuracy on val set:79.714
Accuracy on val set:80.526
Accuracy on val set:81.408
```

```
Accuracy on val set:81.958
Accuracy on val set:82.956
```

```
state_dict = torch.load("best_model.pth")
model.load_state_dict(state_dict, strict=False)
for data in train_loader:
    inputs,target=data
    inputs,target = inputs.to(device), target.to(device)
    outputs=model(inputs)
    _,predicted=torch.max(outputs.data,dim=1)
    total+=target.size(0)
    correct+=(predicted==target).sum().item()
train_acc_1 = 100*correct/total
print('Accuracy on train set:{}'.format(train_acc_1))
for data in val_loader:
    inputs,target=data
    inputs,target = inputs.to(device), target.to(device)
    outputs=model(inputs)
    _,predicted=torch.max(outputs.data,dim=1)
    total+=target.size(0)
    correct+=(predicted==target).sum().item()
val_acc_1 = 100*correct/total
print('Accuracy on val set:{}'.format(val_acc_1))
for data in test_loader:
    inputs,target=data
    inputs,target = inputs.to(device), target.to(device)
    outputs=model(inputs)
    _,predicted=torch.max(outputs.data,dim=1)
    total+=target.size(0)
    correct+=(predicted==target).sum().item()
test_acc_1 = 100*correct/total
print('Accuracy on test set:{}'.format(test_acc_1))
```

```
Accuracy on train set:82.95
Accuracy on val set:82.98133333333334
Accuracy on test set:82.991
```

2.2c Train a neural network composed of **3 convolutional layers and 3 fully connected layers**. Train the model on the training set, use the validation set to choose the best design among **at least three different** choices, and test the chosen model on the test set.

```
# Write your code here.
class net(nn.Module):
    def __init__(self):
        super(net, self).__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(1, 32, 5),
            nn.ReLU(),
            nn.MaxPool2d(2, stride = 2),
            nn.Dropout(0.3),
            nn.Conv2d(32, 64, 5),
            nn.ReLU(),
```

```
            nn.MaxPool2d(2, stride = 2),
            nn.Dropout(0.3),
            nn.Conv2d(64, 32, 1),
            nn.ReLU(),
        )
        self.fc = nn.Sequential(
            nn.Linear(32 * 4 * 4, 512),
            nn.ReLU(),
            nn.Linear(512, 10)
        )
    def forward(self, x):
        x = self.conv(x)
        x = x.view(-1, 32 * 4 * 4)
        x = self.fc(x)
        return x

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = net().to(device)
criterion = torch.nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(),
                      lr=0.001,
                      momentum=0.9,
                      weight_decay=1e-5)
max_acc = 0
for epoch in range(10):
    for idx, (images, labels) in enumerate(train_loader):
        images, labels = images.to(device), labels.to(device)
        output = model(images)
        loss = criterion(output,labels)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    correct=0
    total=0
    with torch.no_grad():
        for data in val_loader:
            inputs,target=data
            inputs,target = inputs.to(device), target.to(device)
            outputs=model(inputs)
            _,predicted=torch.max(outputs.data,dim=1)
            total+=target.size(0)
            correct+=(predicted==target).sum().item()
    acc = 100*correct/total
    print('Accuracy on val set:{}'.format(acc))
    if acc>max_acc:
        max_acc = acc
        torch.save(model.state_dict(),"best_model.pth" )
```

```
Accuracy on val set:55.664
Accuracy on val set:68.432
Accuracy on val set:70.448
Accuracy on val set:74.18
Accuracy on val set:75.936
Accuracy on val set:77.24
```

```
Accuracy on val set:78.554
Accuracy on val set:79.612
Accuracy on val set:80.962
Accuracy on val set:81.206
```

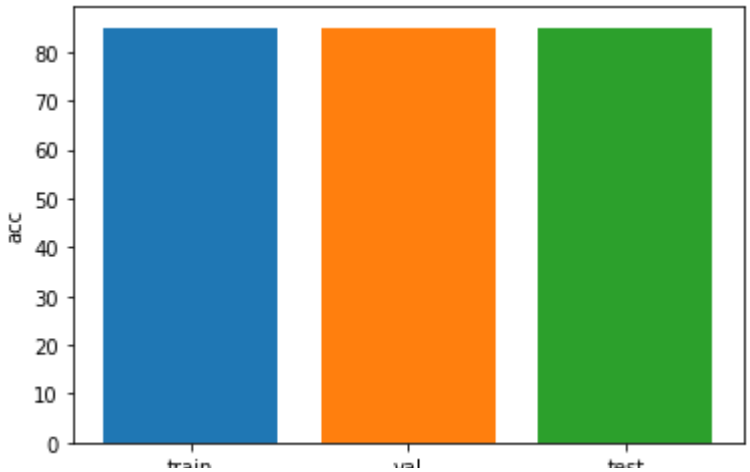
```
state_dict = torch.load("best_model.pth")
model.load_state_dict(state_dict, strict=False)
for data in train_loader:
    inputs,target=data
    inputs,target = inputs.to(device), target.to(device)
    outputs=model(inputs)
    _,predicted=torch.max(outputs.data,dim=1)
    total+=target.size(0)
    correct+=(predicted==target).sum().item()
train_acc_2 = 100*correct/total
print('Accuracy on train set:{}'.format(train_acc_2))
for data in val_loader:
    inputs,target=data
    inputs,target = inputs.to(device), target.to(device)
    outputs=model(inputs)
    _,predicted=torch.max(outputs.data,dim=1)
    total+=target.size(0)
    correct+=(predicted==target).sum().item()
val_acc_2 = 100*correct/total
print('Accuracy on val set:{}'.format(val_acc_2))
for data in test_loader:
    inputs,target=data
    inputs,target = inputs.to(device), target.to(device)
    outputs=model(inputs)
    _,predicted=torch.max(outputs.data,dim=1)
    total+=target.size(0)
    correct+=(predicted==target).sum().item()
test_acc_2 = 100*correct/total
print('Accuracy on test set:{}'.format(test_acc_2))
```

```
Accuracy on train set:81.113
Accuracy on val set:81.072
Accuracy on test set:81.0535
```

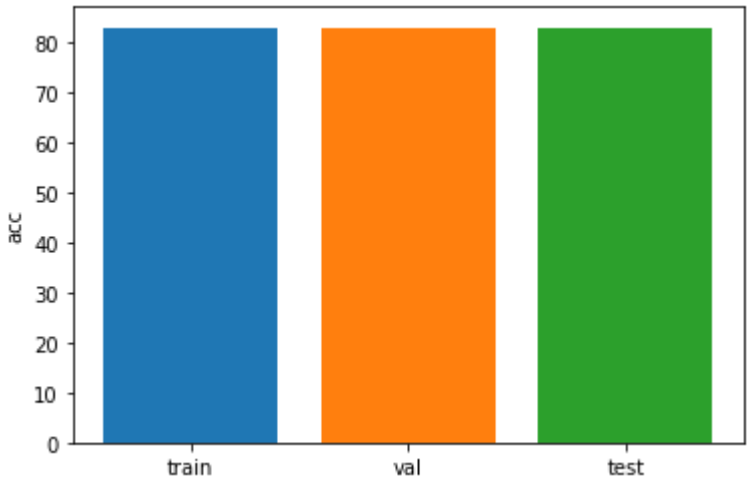
2.3: Comparison of model performance [5 marks]

2.3a In separate **plots**, show the training accuracy, validation accuracy and test accuracy for each of these models.

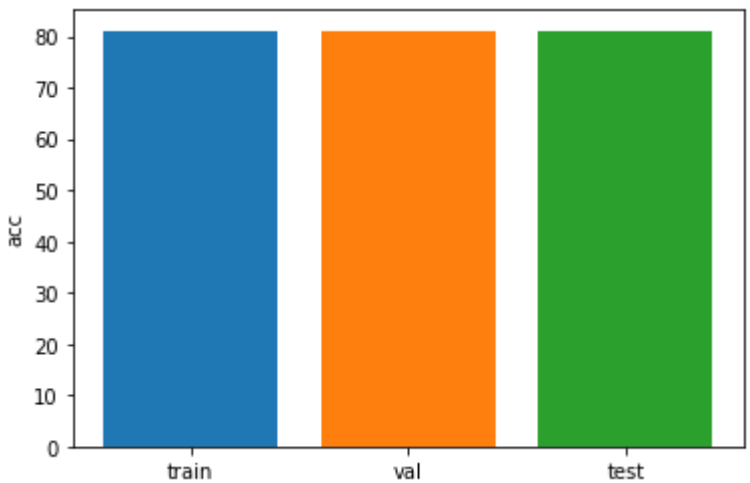
```
# Write your code here.
x_data=['train','val','test']
plt.bar('train', train_acc)
plt.bar('val', val_acc)
plt.bar('test', test_acc)
plt.ylabel("acc")
plt.show()
```




```
plt.bar('train', train_acc_1)
plt.bar('val', val_acc_1)
plt.bar('test', test_acc_1)
plt.ylabel("acc")
plt.show()
```




```
plt.bar('train', train_acc_2)
plt.bar('val', val_acc_2)
plt.bar('test', test_acc_2)
plt.ylabel("acc")
plt.show()
```



2.3b Describe at least **two** observations of the data plotted in this section.

```
# Write your answer here
#Since convolutional neural networks have a large number of parameters, they need t
#Approximately equal accuracy in the training and validation sets and the test set
```

3. Denoising Autoencoder [16 marks]

The CIFAR-10 dataset

In this assignment, we will work on the CIFAR-10 dataset collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton from the University of Toronto. This dataset consists of 60,000 32x32 colour images in 10 classes, with 6,000 images per class. Each sample is a 3-channel colour images of 32x32 pixels in size. There are 50,000 training images and 10,000 test images.

3.1: Data loading and manipulation [3 marks]

3.1a Download both the training and test data of the CIFAR-10 dataset, e.g., by following the pytorch CIFAR10 tutorial. You can also download via other ways if you prefer.

```
# Write your code here.
import torchvision.transforms as transforms
import torchvision.datasets as datasets
import torch.nn as nn
import torch.nn.functional as F
train = datasets.CIFAR10('./', train=True, download=True, transform=transforms.Comp
                                      transforms.Resize(32),
                                      transforms.ToTensor(),
                                      transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0
                                    ]))
test = datasets.CIFAR10('./', train=False, transform=transforms.Compose([
                                      transforms.Resize(32),
                                      transforms.ToTensor(),
                                      transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0
                                    ]))
train_noisy = datasets.CIFAR10('./', train=True, download=True, transform=transform
                                      transforms.Resize(32),
                                      transforms.ToTensor(),
                                      transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0
                                    ]))
test_noisy = datasets.CIFAR10('./', train=False, transform=transforms.Compose([
                                      transforms.Resize(32),
                                      transforms.ToTensor(),
                                      transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0
                                    ]))
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./cifar
100%                                      170498071/170498071 [00:01<00:00, 94327877.33i
Extracting ./cifar-10-python.tar.gz to ./
Files already downloaded and verified
```

3.1b Add random noise to all training and test data to generate noisy dataset, e.g., by torch.randn(), with a scaling factor scale, e.g., original image + scale * torch.randn(), and normalise/standardise the pixel values to the original range, e.g., using np.clip(). You may choose any scale value between 0.2 and 0.5.

A random transformation can be applied using a `Lambda` transform when composing the load data transform, which looks a little like this:

```
transforms.Lambda(lambda x: x + ..... )
```

Note: Before generating the random noise, you MUST set the random seed to your UCard number XXXXXXXXX for reproducibility, e.g., using torch.manual_seed(). This seed needs to be used for all remaining code if there is randomness, for reproducibility.

You may want to create separate dataloaders for the noisy and clear images but make sure they are not shuffling the data so that correct pair of images are being given as input and desired output.

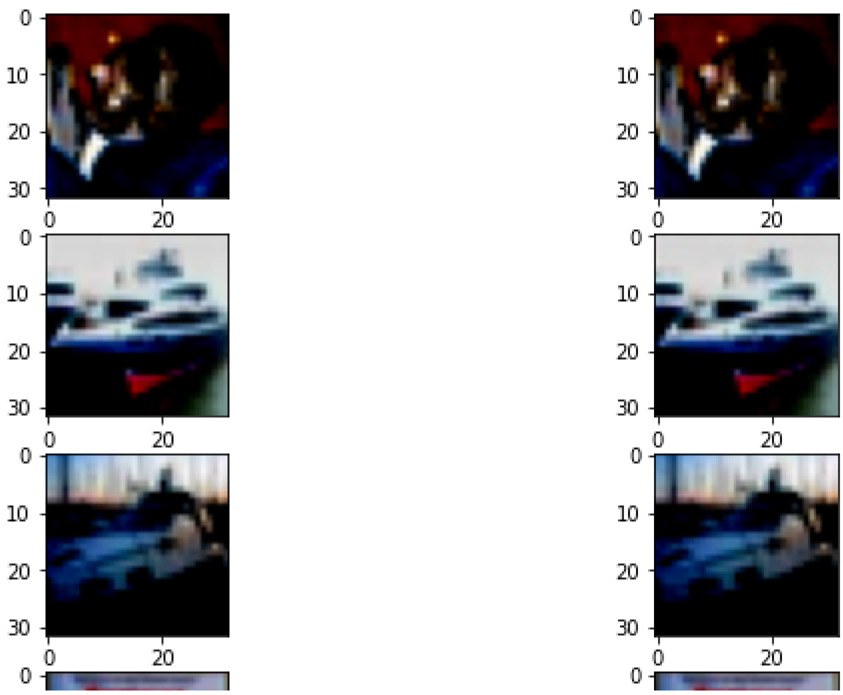
```
# Write your code here.
import numpy as np
torch.manual_seed(42)

# train_noisy = train
# test_noisy = test
train_noisy.data = np.uint8(np.clip(train_noisy.data+0.2*torch.randn(50000,32,32,3)
test_noisy.data = np.uint8(np.clip(test_noisy.data + 0.2*torch.randn(10000,32,32,3)

# print(noisy.shape)
# print(train_noisy.data)
# print(train.data)
train_loader = torch.utils.data.DataLoader(train, batch_size=64,
    shuffle=True, pin_memory=True)
test_loader = torch.utils.data.DataLoader(test, batch_size=1,
    shuffle=False)
train_noisy_loader = torch.utils.data.DataLoader(train_noisy, batch_size=64,
    shuffle=True, pin_memory=True)
test_noisy_loader = torch.utils.data.DataLoader(test_noisy, batch_size=1,
    shuffle=False)
```

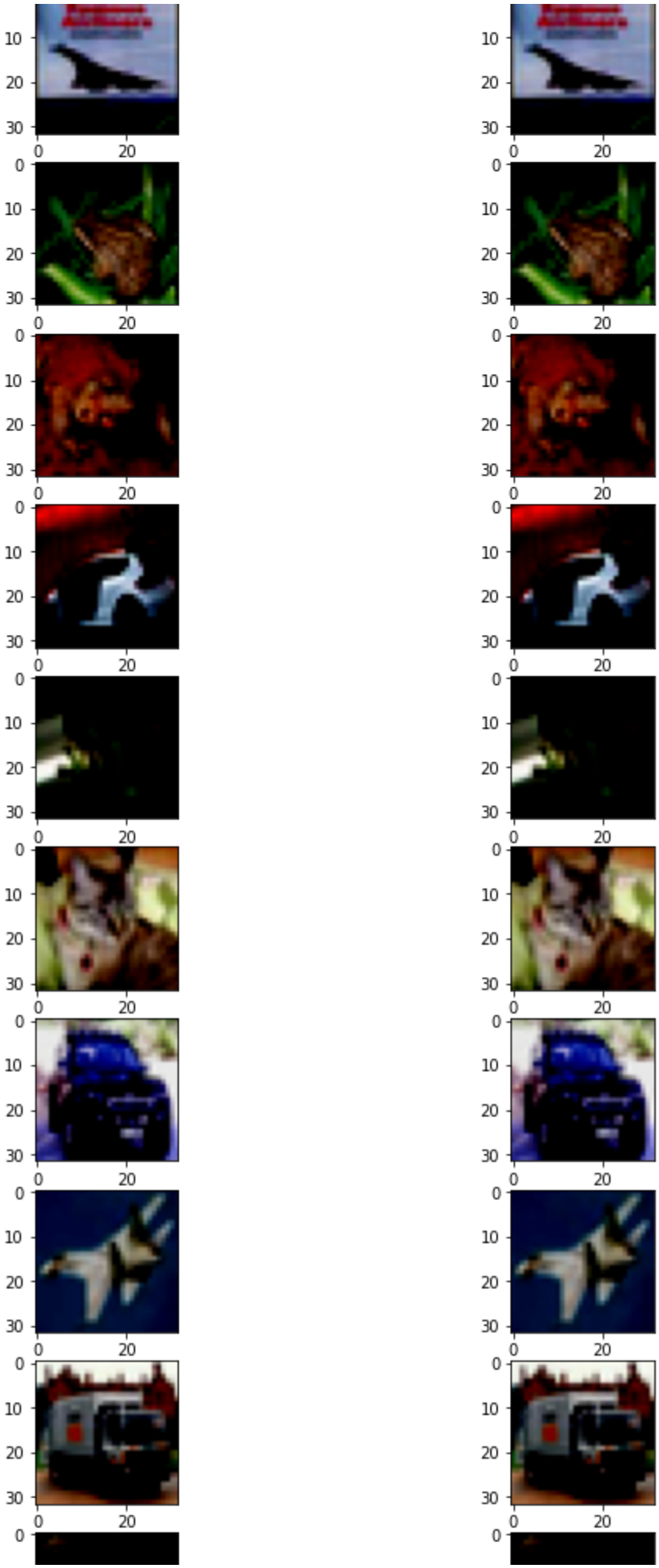
3.1c Show 20 pairs of original and noisy images.

```
# Write your code here.
r,c=20,2
w,h=32,32
fig,axs=plt.subplots(r,c,figsize=(10,40))
for i,(image,label) in enumerate(test_loader):
```

```
        axs[i,0].imshow(np.transpose(image[0],(1,2,0)))
        if i ==19:
            break
    for i,(image,label) in enumerate(test_noisy_loader):
        axs[i,1].imshow(np.transpose(image[0],(1,2,0)))
        if i ==19:
            break
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
```





3.2 Applying a Denoising Autoencoder to the modified CIFAR10 [10 marks]

This question uses both the original and noisy CIFAR-10 datasets (all 10 classes). Read about denoising autoencoders at Wikipedia and this short introduction or any other sources you like.


3.2a Modify the autoencoder architecture in Lab 8 so that it takes colour images as input (i.e., 3 input channels).


```
# Write your code here.
"""
Here, we define the autoencoder model.
"""

class denoising_model(nn.Module):
    def __init__(self):
        super(denoising_model,self).__init__()
        self.encoder=nn.Sequential(
                     nn.Linear(32*32*3,256),
                     nn.ReLU(True),
                     nn.Linear(256,128),
                     nn.ReLU(True),
                     nn.Linear(128,64),
                     nn.ReLU(True)

                     )

        self.decoder=nn.Sequential(
                     nn.Linear(64,128),
                     nn.ReLU(True),
                     nn.Linear(128,256),
                     nn.ReLU(True),
                     nn.Linear(256,32*32*3),
                     nn.Sigmoid(),
                     )
    def forward(self,x):
        x=self.encoder(x)
        x=self.decoder(x)

        return x
```


3.2b Training: feed the noisy training images as input to the autoencoder defined above; use a loss function that computes the reconstruction error between the output of the autoencoder and the respective original images.

```
# Write your code here.
model=denoising_model().to(device)
criterion=nn.MSELoss()
optimizer=optim.SGD(model.parameters(),lr=0.01,weight_decay=1e-5)


epochs=10
l = len(train_loader)
```

```
l = len(train_loader)
losslist=list()
epochloss=0
running_loss=0
for epoch in range(epochs):
    for (dirty,label),(clean,_) in zip(train_noisy_loader,train_loader):

        dirty=dirty.view(dirty.size(0),-1).type(torch.FloatTensor)
        clean=clean.view(clean.size(0),-1).type(torch.FloatTensor)
        dirty,clean=dirty.to(device),clean.to(device)
    #-----------------Forward Pass----------------------
        output=model(dirty)
        loss=criterion(output,clean)
    #-----------------Backward Pass---------------------
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        running_loss+=loss.item()
        epochloss+=loss.item()
    losslist.append(running_loss/l)
    running_loss=0
    print("======> epoch: {}/{}, Loss:{}".format(epoch,epochs,loss.item()))
```

```
======> epoch: 0/10, Loss:0.539165198802948
======> epoch: 1/10, Loss:0.6595542430877686
======> epoch: 2/10, Loss:0.5996041893959045
======> epoch: 3/10, Loss:0.49800631403923035
======> epoch: 4/10, Loss:0.5329141020774841
======> epoch: 5/10, Loss:0.571878969669342
======> epoch: 6/10, Loss:0.569599986076355
======> epoch: 7/10, Loss:0.5046303868293762
======> epoch: 8/10, Loss:0.5119368433952332
======> epoch: 9/10, Loss:0.4763350784778595
```

3.2c Testing: evaluate the autoencoder trained in 3.2b on the test datasets (feed noisy images in and compute reconstruction errors on original clean images. Find the worst denoised 30 images (those with the largest reconstruction errors) in the test set and show them in pairs with the original images (60 images to show in total).

```
# Write your code here.
l = len(test_loader)
loss_total = 0
L = []
for (dirty,label),(clean,_) in zip(test_noisy_loader,test_loader):

    dirty=dirty.view(dirty.size(0),-1).type(torch.FloatTensor)
    clean=clean.view(clean.size(0),-1).type(torch.FloatTensor)
    dirty,clean=dirty.to(device),clean.to(device)
#-----------------Forward Pass----------------------
    output=model(dirty)
    loss=criterion(output,clean)
    L.append(loss.item())
```

```
        loss_total+=loss.item()
    print(loss_total/l)
```

```
     0.5489281444706022
```

```
    index = np.argsort(L)[::-1]
    index = index[:30]
    print(index)
```

```
     [9246 6941 6885 6234 2754 9765 8244 3119  872 8448 3582 7794 8381  118
      4550 5927 3758 1300 1240 8595 2940  907 6555 9344 8835 3724 4355 8768
      4478 5671]
```

```
    r,c=30,2
    w,h=32,32
    fig,axs=plt.subplots(r,c,figsize=(20,60))
    num=0
    for i,(image,label) in enumerate(test_loader):
        if i in index:
            axs[num,0].imshow(np.transpose(image[0],(1,2,0)))
            num+=1
        if num==30:
            break
    num=0
    for i,(image,label) in enumerate(test_noisy_loader):
        if i in index:
            axs[num,1].imshow(np.transpose(image[0],(1,2,0)))
            num+=1
        if num ==30:
            break
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
WARNING:matplotlib.image:Clipping input data to the valid range for imshow wit
```