Name: Quang Dong Nguyen
Student ID: 20744696
Class: Machine Learning
University: Western Sydney University

Assignment 2

About CIFAR:
The CIFAR-10 dataset is a collection of images that are commonly used to train machine learning and computer vision algorithms. It is one of the most widely used datasets for machine learning research. The CIFAR-10 dataset contains 60,000 32x32 color images in 10 different classes. The 10 different classes represent airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. There are 6,000 images of each class. See https://www.cs.toronto.edu/~kriz/cifar.html for details

Task:

1. Randomly select 3 classes with 100 images per class for this assignment;
2. Build the autoencoder model using CNN with functioning training code (if not CNN based, 60% reduction of marks will incur for this task);
3. Plot the learnt images 2D coordinates (normally called *embeddings* in machine learning) of all images in training with each class denoted b a symbol, for example, circles for dogs, triangles for cats and so on;
4. Randomly select 5 images that are not in the training set and obtain their 2D representations, add them to the plot produced in task 3 and describe what do you think about them in terms of their locations in relations to others. Your code should produce the plot similar to fig 1

Bonus Task:

Build a **supervised** manifold learning model on CIFAR-10 images. The main idea is to incorporate labels information in the manifold learning process. It is very similar to LDA (linear discriminant analysis) in terms of functionality. However, instead of a lienar function, we use neural networks autoencoder as the backbone for manifold learning. Therefore, the model is a combination of autoencoder and classification, i.e. incorporating supervision information inthe modelling process, for exmaple, addding classification cost function into original autoencoder cost function. Do task 1-4 (see above) but replace the autoencoder by this supervised one.
NOTE: this is extra 10 marks contributing towards your final scores if you can do it

---

Firstly, we import the library that will be used in this assignment

In [52]:
```
import numpy as np
import torch
import torchvision
import os
import random
import matplotlib.lines as mlines
import matplotlib.pyplot as plt
import pandas as pd

#random split modules
from torch.utils.data import random_split
from torch.utils.data import DataLoader

#to load and normalise CIFAR10
from torchvision.datasets import CIFAR10
from torchvision import transforms

#neural network training
from torch import nn
from torch.nn import ConvTranspose2d
```

Question 1:

Firstly, we load the image dataset from CIFAR10 packages

In [53]:
```
#loading the dataset
torch.manual_seed(0)
CIF_dataset = CIFAR10(os.getcwd(), transform = transforms.ToTensor(), download = False) #CIFAR dataset
classes = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truch']
```

We then randomly choose 3 classes from this dataset that will be used for further analysis

- From the random choice below, we got 3 classes: 2, 1, 8, which stands for bird, automobile, and ship

In [54]:
```
#choosing 3 random images classes:
random.seed(1)
random.sample(range(0,9), 3) #this return 2, 1, 8, which stand for bird, automobile and ship
```

Out[54]: `[2, 1, 8]`

Then, from the dataset, the 3 classes mentioned are selectively sorted out for their indices and binded into a group

In [55]:
```
#From CIFAR-10 dataset, we select 100 images for each of the 3 classes:
row_1 = list(np.where(np.array(CIF_dataset.targets) == 2)[0])[0:100]
row_2 = list(np.where(np.array(CIF_dataset.targets) == 1)[0])[0:100]
row_3 = list(np.where(np.array(CIF_dataset.targets) == 8)[0])[0:100]
row_binded = row_1 + row_2 + row_3
```

```
for i in range(10):
    random.Random(i).shuffle(row_binded) #shuffled 10 times
```

Using the group of indices created above to create a subset of values of images.

In [56]:
```
#use subset to bind rows that matched:
CIF_dataset1 = torch.utils.data.Subset(CIF_dataset, row_binded)
#len(CIF_dataset) #300 images for 3 classes (100 images per class)
#CIF_dataset.shape
```

Then we split the data into training set, validation set, and test set, each is provided with 100 images.

In [57]:
```
#Create sets for training, valdating and testing
train_dataset, val_dataset, test_dataset = torch.utils.data.random_split(CIF_dataset1, [100,100,100])

#Load data
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size = 5, shuffle = True)
val_loader = torch.utils.data.DataLoader(val_dataset, batch_size = 5, shuffle = True)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size = 5, shuffle = False)
```

Question 2:

On this step we create a CNN AutoEncoder having the bottleneck layer output as 2 units.

Based on some researched information, the layer size can be determined as follows:

- CONVOLUTIONAL LAYER : (W - F + 2P)/S + 1
- TRANSPOSED CONVOLUTIONAL LAYER: (W - 1) * S - 2P + F
- W = input size (default = 0)
- F = kernel_size (default = 0)
- P = padding (default = 0)
- S = stride (default = 1)

In [58]:
```
#++++++++++++++++++++
#CNN AUTOENCODER
class CNNAutoEncoder(nn.Module):
    def __init__(self):
        super().__init__()
        #N, 3, 32 * 32 size
        self.encoder_layer = nn.Sequential(
            # Conv_layer block 1
            nn.Conv2d(in_channels = 3, out_channels = 32, kernel_size = 5, padding = 2), #32 x 32 x 32
            nn.BatchNorm2d(32),
            nn.ReLU(inplace = True),

            #Conv_layer block 2
            nn.Conv2d(in_channels = 32, out_channels = 16, kernel_size = 3, padding = 1), # 16 x 32 x 32
            nn.ReLU(inplace = True),
            nn.MaxPool2d(kernel_size = 2, stride = 2), #16 x 16 x 16

            #Conv_layer block 3
            nn.Conv2d(in_channels = 16, out_channels = 16, kernel_size = 5, padding = 2), # 16 x 16 x 16
            nn.ReLU(inplace = True),
            nn.MaxPool2d(kernel_size = 2, stride = 2), #16 x 8 x 8

            #Conv_layer block 4
            nn.Conv2d(in_channels = 16, out_channels = 8, kernel_size = 3, padding = 1), # 8 x 8 x 8
            nn.ReLU(inplace = True),
            )


        self.bottle_neck = nn.Sequential(
            nn.Linear(8 * 8 * 8, 2) # there are 2 units as output
            )


        self.decoder_layer = nn.Sequential(
            #Deconv_layer block 1
            nn.ConvTranspose2d(in_channels = 8, out_channels = 16, kernel_size = 2, stride = 2), #16 x 16 x 16
            nn.ReLU(inplace = True),

            #Deconv_Layer block 2
            nn.ConvTranspose2d(in_channels = 16, out_channels = 32, kernel_size = 2, stride = 2), # 32 x 32 x 32
            nn.ReLU(inplace = True),

            #Deconv_Layer block 3
            nn.ConvTranspose2d(in_channels = 32, out_channels = 3, kernel_size = 3, padding = 1), #3 x 32 x 32 #the output is the same as the input
            nn.ReLU(inplace = True)
            )


    def forward(self, x):
        #encoder_layer
        encoder_val = self.encoder_layer(x)

        #bottleneck_layer
        bottle_neck = encoder_val.view(encoder_val.size(0), - 1) #in terms of dimensionality, we fix it to be 2
        bottle_neck = self.bottle_neck(bottle_neck)

        #decoder_layer
        x = self.decoder_layer(encoder_val)

        return bottle_neck, x
```

```
model = CNNAutoEncoder()
#+++++++++++++++++++
```

And then we move onto creating a function to help train the CNN AutoEncoder model declared above, the required values to be inserted into the function will mainly consist of:

- Optimiser
- Chosen model
- Loss function
- Train dataloader
- Validation dataloader
- number of epochs
- etc

In [59]:
```
#+++++++++++++++++++
def AutoEncoder_model_training(optimiser, model,
                               Loss_function,
                               trainloader, valloader,
                               n_epochs = 10, fplotloss = True, #fdraw = False,
                               filename = ''):
    train_on_gpu = torch.cuda.is_available()
    if train_on_gpu:
        print("GPU available! Train model on GPU.")
        model.cuda()

    #tracking
    train_LossList = []
    val_LossList = []

    val_Loss_Min = np.Inf

    #Entering Training Cycles
    print("Entering training cycles with CNN AutoEncoder")
    for epoch in [*range(n_epochs)]:

        #keeping tacking of training loss and validation loss
        train_Loss = 0.0
        val_Loss = 0.0


        #for train model
        model.train()
        for data, target in trainloader:
            if train_on_gpu:
                data, target = data.cuda(), target.cuda()

            #clear gradient of all optimised variables
            optimiser.zero_grad()

            #forward pass: predicted outputs by passing inputs to the model
            output = model(data)

            #batch loss:
            Batch_Loss = Loss_function(output[1], data)

            #backward pass: compute gradients of the loss with respect to model parameters
            Batch_Loss.backward()

            #optimisation step (parameter update)
            optimiser.step()

            #update training Loss
            train_Loss += Batch_Loss.item() * data.size(0)

        #validate the model
        model.eval()
        for data, target in valloader:
            if train_on_gpu:
                data, target = data.cuda(), target.cuda()

            #forward pass: predicts outputs by passing inputs to the model
            output = model(data)

            #batch_loss:
            Batch_Loss = Loss_function(output[1], data)

            #Update validation loss:
            val_Loss += Batch_Loss.item() * data.size(0)

        #Calculate avg losses
        train_Loss = train_Loss / len(trainloader.dataset)
        val_Loss = val_Loss / len(valloader.dataset)

        #append the Loss values to the Loss lists declared
        train_LossList.append(train_Loss)
        val_LossList.append(val_Loss)

        #print the statistics
        print('Epoch: {} \tTraining_Loss: {:.6f} \t Validation_Loss: {:.6f}'.format(epoch, train_Loss, val_Loss))

        #if validation loss decreased
        if val_Loss <= val_Loss_Min: #print if val loss decreased
            print('Validation Loss decreased: ({:.6f} --> {:.6f}). Saving.'.format(val_Loss_Min, val_Loss))
            torch.save(model.state_dict(), 'bestCNNAutoEncoder_model' + filename + '.pt') #we then proceed onto saving the best model
            val_Loss_Min = val_Loss
```

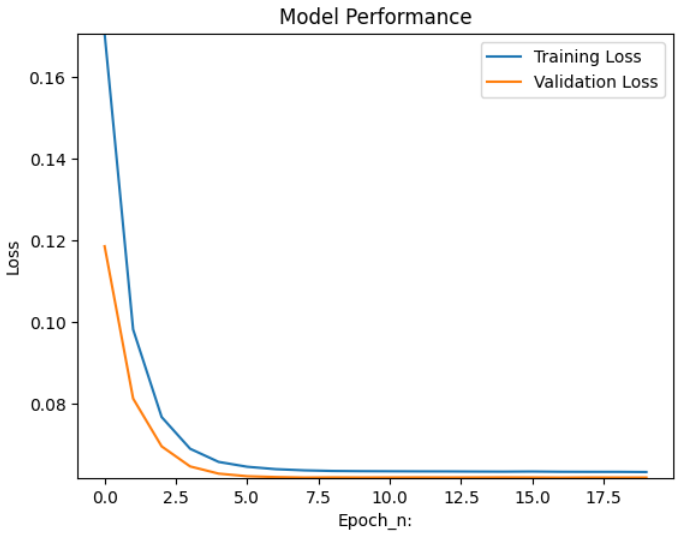
```
    #Plot Training and Validation Loss if fplotloss = True
    if fplotloss == True:
        plt.plot(*range(n_epochs), train_LossList)
        plt.plot(*range(n_epochs), val_LossList)
        plt.ylim((min(train_LossList + val_LossList), max(train_LossList + val_LossList)))
        plt.xlabel('Epoch_n:')
        plt.ylabel('Loss')
        plt.title('Model Performance')
        plt.legend(['Training Loss', 'Validation Loss'])
        plt.show()

    #Cycles is complete
    print('Training process is now completed')
#+++++++++++++++++++
```

We are then moving on to training the model:

- Since with this problem, we want to train the model until it becomes the best model, so that we can use that model to visualise the images from the bottleneck layer of the CNN AutoEncoder model above onto a 2D coordinate graph. We will train the model as follows:

In [60]:
```
#train the model to the best model
model = CNNAutoEncoder()
optimiser = torch.optim.SGD(model.parameters(), lr = 0.01) #We use stochastic gradient descent as an optimiser for this model

#train the model to the best model
AutoEncoder_model_training(optimiser, model,
                           nn.MSELoss(), #Mean Squared Error Loss
                           train_loader, val_loader,
                           n_epochs = 20, fplotloss = True, filename = "_ver1")
```

```
Entering training cycles with CNN AutoEncoder
Epoch: 0        Training_Loss: 0.170646         Validation_Loss: 0.118537
Validation Loss decreased: (inf --> 0.118537). Saving.
Epoch: 1        Training_Loss: 0.098136         Validation_Loss: 0.081209
Validation Loss decreased: (0.118537 --> 0.081209). Saving.
Epoch: 2        Training_Loss: 0.076679         Validation_Loss: 0.069489
Validation Loss decreased: (0.081209 --> 0.069489). Saving.
Epoch: 3        Training_Loss: 0.068900         Validation_Loss: 0.064567
Validation Loss decreased: (0.069489 --> 0.064567). Saving.
Epoch: 4        Training_Loss: 0.065675         Validation_Loss: 0.062804
Validation Loss decreased: (0.064567 --> 0.062804). Saving.
Epoch: 5        Training_Loss: 0.064490         Validation_Loss: 0.062168
Validation Loss decreased: (0.062804 --> 0.062168). Saving.
Epoch: 6        Training_Loss: 0.063901         Validation_Loss: 0.061965
Validation Loss decreased: (0.062168 --> 0.061965). Saving.
Epoch: 7        Training_Loss: 0.063610         Validation_Loss: 0.061848
Validation Loss decreased: (0.061965 --> 0.061848). Saving.
Epoch: 8        Training_Loss: 0.063459         Validation_Loss: 0.061883
Epoch: 9        Training_Loss: 0.063406         Validation_Loss: 0.061860
Epoch: 10       Training_Loss: 0.063382         Validation_Loss: 0.061872
Epoch: 11       Training_Loss: 0.063358         Validation_Loss: 0.061884
Epoch: 12       Training_Loss: 0.063346         Validation_Loss: 0.061868
Epoch: 13       Training_Loss: 0.063314         Validation_Loss: 0.061874
Epoch: 14       Training_Loss: 0.063292         Validation_Loss: 0.061887
Epoch: 15       Training_Loss: 0.063326         Validation_Loss: 0.061873
Epoch: 16       Training_Loss: 0.063260         Validation_Loss: 0.061834
Validation Loss decreased: (0.061848 --> 0.061834). Saving.
Epoch: 17       Training_Loss: 0.063242         Validation_Loss: 0.061862
Epoch: 18       Training_Loss: 0.063236         Validation_Loss: 0.061851
Epoch: 19       Training_Loss: 0.063191         Validation_Loss: 0.061846
```




```
Training process is now completed
```

Load the best model from all the trained model

In [61]:
```
#load the best trained model:
model.load_state_dict(torch.load(r'D:\dong;s junior (WSU)\Second Year - 2023\Semester 2\Machine Learning\Assignment\Assignment 2 - due 12 Nov\bestCN
```

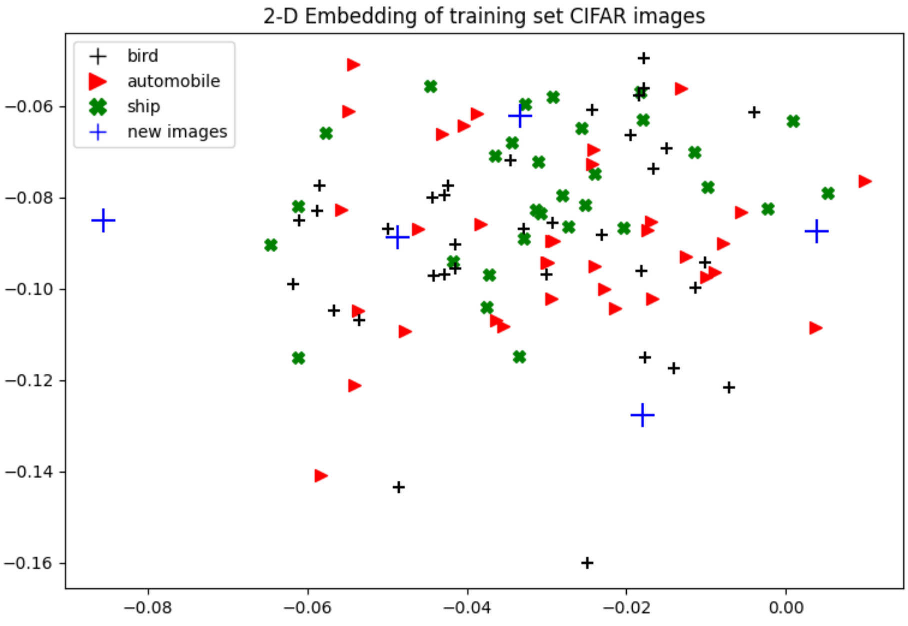
Out[61]: <All keys matched successfully>

Question 3 and 4:

With this question, we will use the output produced by the bottle neck layer of the best model as X and y coordinates for each of the images from the training set, and plot it to a 2D graph. Furthermore, from the `test_loader` dataset, we will plot 5 images coordinates after having it trained through the loaded best model above.

In [62]:
```
#Create markers for each input of images
X_sym = {2: '+', 1: '>' , 8: 'X'}
X_col = {2: 'black', 1: 'red', 8:'green'}
marker_1 = mlines.Line2D([],[], color = 'black', marker = '+', linestyle = 'None', markersize = 10, label = 'bird')
marker_2 = mlines.Line2D([],[], color = 'red', marker = '>', linestyle = 'None', markersize = 10, label = 'automobile')
marker_3 = mlines.Line2D([],[], color = 'green', marker = 'X', linestyle = 'None', markersize = 10, label = 'ship')
marker_4 = mlines.Line2D([],[], color = 'blue', marker = '+', linestyle = 'None', markersize = 10, label = 'new images')


plt.figure(figsize = (9,6))
#for plotting dataset from the train_loader:
for a in train_loader:
    lbls = a[1]
    output_btn, x_decoded = model(a[0])
    output_plot = pd.DataFrame(data = output_btn.detach().numpy())
    output_plot['labels'] = lbls.detach().numpy()
    output_plot['color'] = output_plot['labels'].replace(X_col)
    output_plot['symbols'] = output_plot['labels'].replace(X_sym)
    for i in range(len(output_plot)):
        plt.scatter(output_plot[0][i], output_plot[1][i], c = output_plot['color'][i], marker = output_plot['symbols'][i], s = 50)

plt.title('2-D Embedding of training set CIFAR images')

#for plotting the 5 new images from the test_loader:
i = 0
for a in test_loader:
    if i <= 1:
        lbls = a[1]
        output_btn, x_decoded = model(a[0])
        output_plot = pd.DataFrame(data = output_btn.detach().numpy())
        output_plot['labels'] = lbls.detach().numpy()
        output_plot['color'] = 'blue'
        output_plot['symbols'] = '+'
        for i in range(len(output_plot)):
            plt.scatter(output_plot[0][i], output_plot[1][i], c = output_plot['color'][i], marker = output_plot['symbols'][i], s = 200)


#finally we add in the legend for the plot
plt.legend(handles = [marker_1, marker_2, marker_3, marker_4])
```

Out[62]: `<matplotlib.legend.Legend at 0x1d6b5b845b0>`



In terms of location of the 5 new images:

- The 1st image with the coordinate of x, y: -0.09, -0.08, it is located on the far left side of the graph where it could belong to any of the 3 classes of images.
- However, the 2nd image's coordinate (x, y: -0.05, -0.09), it could potentially belong to the bird class since the surrounding coordinates are mostly from bird class and only some are from the automobile class.
- With the 3rd image's coordinate, it is more toward the middle top of the graph, where there are many clusters of ship coordinate points gathered over its place. Hence it is more persuasive to assume that this image belongs to the ship class.
- Fourthly, the image with the coordinate of -0.02 on x and -0.12 on y, locates on the middle of graph where the surrounding classes are mostly bird, and automobile. Therefore, it could be either of the 2 classes.
- Lastly, the last point of image (x, y: 0.01, -0.08), it could be either automobile class or ship class since the closest classes are these 2 classes.

Overall, each point of the 5 new images is unique and distinct from each other in terms of locations. However, these are just assumptions made purely based on the insight given by the visualisation above, further investigation will be required to assess the labels of the 5 images.

The Bonus Part:

In this part, we create a supervised manifold learning model, where it will be used as both autoencoder and classification model. However, firstly, we need to redefine the label class for this dataset. Since we are only predicting 3 classes, the total classes we should have are 3 classes and each should be labelled consecutively. For instance, 0: ship, 1: automobile and 2: bird. Otherwise, it will not accurately predict the accuracy score later on for the model.

In [63]:
```
#Loading the dataset
torch.manual_seed(0)
CIF_dataset = CIFAR10(os.getcwd(), transform = transforms.ToTensor(), download = False) #CIFAR dataset
CIF_dataset.targets[CIF_dataset.targets == 0] = 11 #airplane is now labelled as 11 #we will not use the airplane dataset
CIF_dataset.targets[CIF_dataset.targets == 8] = 0 #ship is now labelled as 0
#classes = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truch']
```

Again, we then bind the row of indices together

In [64]:
```
#From CIFAR-10 dataset, we select 100 images for each of the 3 classes:
row_1 = list(np.where(np.array(CIF_dataset.targets) == 2)[0])[0:100]
row_2 = list(np.where(np.array(CIF_dataset.targets) == 1)[0])[0:100]
row_3 = list(np.where(np.array(CIF_dataset.targets) == 0)[0])[0:100]
row_binded = row_1 + row_2 + row_3

for i in range(10):
    random.Random(i).shuffle(row_binded) #shuffled 10 times
```

We create a new subset of dataset which consists only ship, automobile and bird. And the label for ship is 0, automobile as 1 and bird as 2

In [65]:
```
#use subset to bind rows that matched:
CIF_dataset1 = torch.utils.data.Subset(CIF_dataset, row_binded)
#len(CIF_dataset1) #300 images for 3 classes (100 images per class)
```

We split the `CIF_dataset1` into training set, validation set and test set for the model

In [66]:
```
#Create sets for training, valdating and testing
train_dataset, val_dataset, test_dataset = torch.utils.data.random_split(CIF_dataset1, [100,100,100])

#Load data
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size = 5, shuffle = True)
val_loader = torch.utils.data.DataLoader(val_dataset, batch_size = 5, shuffle = True)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size = 5, shuffle = False)
```

We then move on creating a supervised learning model which will be used to classify images'labels

In [67]:
```
#++++++++++++++++++++
#CNN AUTOENCODER
class CNNAutoEncoderCls(nn.Module):
    def __init__(self):
        super().__init__()

        #N, 3, 32 * 32 size
        self.encoder_layer = nn.Sequential(
            # Conv_layer block 1
            nn.Conv2d(in_channels = 3, out_channels = 32, kernel_size = 5, padding = 2), #32 x 32 x 32
            nn.BatchNorm2d(32),
            nn.ReLU(inplace = True),

            #Conv_layer block 2
            nn.Conv2d(in_channels = 32, out_channels = 16, kernel_size = 3, padding = 1), #16 x 32 x 32
            nn.ReLU(inplace = True),

            #Conv_layer block 3
            nn.Conv2d(in_channels = 16, out_channels = 16, kernel_size = 5, padding = 2), #16 x 32 x 32
            nn.ReLU(inplace = True),
            nn.MaxPool2d(kernel_size = 2, stride = 2), #16 x 16 x 16

            #Conv_layer block 4
            nn.Conv2d(in_channels = 16, out_channels = 8, kernel_size = 3, padding = 1), #8 x 16 x 16
            nn.ReLU(inplace = True),
            nn.MaxPool2d(kernel_size = 2, stride = 2) #8 x 8 x 8
            )


        self.fc_layer = nn.Sequential(
            nn.Dropout(p = 0.1),
            nn.Linear(8 * 8 * 8, 128),
            nn.ReLU(inplace = True),
            nn.Linear(128, 64),
            nn.ReLU(inplace = True),
            nn.Linear(64, 32),
            nn.ReLU(inplace = True),
            nn.Dropout(p = 0.07),
            nn.Linear(32, 3), #there are 3 classes in total
            nn.Softmax(dim = 1)
            )


    def forward(self, x):
        #encoder_layer
        encoder_val = self.encoder_layer(x)

        #flatten
        encoder_val = encoder_val.view(encoder_val.size(0), - 1)
```

```
        #fc_layer
        encoder_val = self.fc_layer(encoder_val)

        return encoder_val
#+++++++++++++++++++
```

The code below is a training model used to train to classify images'labels. It will require some of the inputs such as:

- optimiser
- model
- loss function parameter
- train dataloader
- validation dataloader
- number of epochs for the training model
- etc (as mentioned in the section below)

In [68]:
```
#+++++++++++++++++++
def AutoEncoderCls_model_training(optimiser, model,
                               Loss_function,
                               trainloader, valloader,
                               n_epochs = 10, fplotloss = True, #fdraw = False,
                               filename = ''):
    train_on_gpu = torch.cuda.is_available()
    if train_on_gpu:
        print("GPU available! Train model on GPU.")
        model.cuda()

    #tracking
    train_LossList = []
    val_LossList = []

    val_Loss_Min = np.Inf

    #Entering Training Cycles
    print("Entering training cycles with CNN AutoEncoder")
    for epoch in [*range(n_epochs)]:

        #keeping tacking of training loss and validation loss
        train_Loss = 0.0
        val_Loss = 0.0


        #for train model
        model.train()
        for data, target in trainloader:
            if train_on_gpu:
                data, target = data.cuda(), target.cuda()

            #clear gradient of all optimised variables
            optimiser.zero_grad()

            #forward pass: predicted outputs by passing inputs to the model
            output = model(data)

            #batch loss:
            Batch_Loss = Loss_function(output, target)

            #backward pass: compute gradients of the loss with respect to model parameters
            Batch_Loss.backward()

            #optimisation step (parameter update)
            optimiser.step()

            #update training Loss
            train_Loss += Batch_Loss.item() * data.size(0)

        #validate the model
        model.eval()
        for data, target in valloader:
            if train_on_gpu:
                data, target = data.cuda(), target.cuda()
            #forward pass: predicts outputs by passing inputs to the model
            output = model(data)
            #batch_loss:
            Batch_Loss = Loss_function(output, target)
            #Update validation loss:
            val_Loss += Batch_Loss.item() * data.size(0)

        #Calculate avg losses
        train_Loss = train_Loss / len(trainloader.dataset)
        val_Loss = val_Loss / len(valloader.dataset)

        #append the Loss values to the Loss lists declared
        train_LossList.append(train_Loss)
        val_LossList.append(val_Loss)

        #print the statistics
        print('Epoch: {} \tTraining_Loss: {:.6f} \t Validation_Loss: {:.6f}'.format(epoch, train_Loss, val_Loss))

        #if validation loss decreased
        if val_Loss <= val_Loss_Min: #print if val loss decreased
            print('Validation Loss decreased: ({:.6f} --> {:.6f}). Saving..'.format(val_Loss_Min, val_Loss))
            torch.save(model.state_dict(), 'bestCNNAutoEncoder_Cls' + filename + '.pt')
            val_Loss_Min = val_Loss
```

```
    #Plot Training and Validation Loss if fplotloss = True
    if fplotloss == True:
        plt.plot(*range(n_epochs), train_LossList)
        plt.plot(*range(n_epochs), val_LossList)
        plt.ylim((min(train_LossList + val_LossList), max(train_LossList + val_LossList)))
        plt.xlabel('Epoch_n:')
        plt.ylabel('Loss')
        plt.title('Model Performance')
        plt.legend(['Training Loss', 'Validation Loss'])
        plt.show()

    #Cycles is complete
    print('Training process is now completed')
#+++++++++++++++++++
```

We are then create an accuracy score function to calculate the overall accuracy score of the model on classifying images' labels

In [69]:
```
#++++++++++++++++++++++++
#accuracy
def Accuracy_score(y_pred, y):
#y_pred: predicted output
#y: true labels
    top_y_pred = y_pred.argmax(1, keepdim = True)
    correct = top_y_pred.eq(y.view_as(top_y_pred)).sum()
    accuracy_score = correct.float() / y.shape[0]
    return accuracy_score, top_y_pred

#+++++++++++++++++++++++++
```

And then we train the model with training set and validation set

In [70]:
```
#torch.manual_seed(42)
model = CNNAutoEncoderCls()
optimiser = torch.optim.SGD(model.parameters(), lr = 0.1)

AutoEncoderCls_model_training(optimiser = optimiser, model = model, Loss_function = nn.CrossEntropyLoss(),
                              trainloader = train_loader, valloader = val_loader,
                              n_epochs = 25, fplotloss = False, filename = '_ver1')
```

```
Entering training cycles with CNN AutoEncoder
Epoch: 0        Training_Loss: 1.098314         Validation_Loss: 1.097691
Validation Loss decreased: (inf --> 1.097691). Saving..
Epoch: 1        Training_Loss: 1.098257         Validation_Loss: 1.097641
Validation Loss decreased: (1.097691 --> 1.097641). Saving..
Epoch: 2        Training_Loss: 1.097443         Validation_Loss: 1.097585
Validation Loss decreased: (1.097641 --> 1.097585). Saving..
Epoch: 3        Training_Loss: 1.097215         Validation_Loss: 1.097527
Validation Loss decreased: (1.097585 --> 1.097527). Saving..
Epoch: 4        Training_Loss: 1.097614         Validation_Loss: 1.097480
Validation Loss decreased: (1.097527 --> 1.097480). Saving..
Epoch: 5        Training_Loss: 1.097130         Validation_Loss: 1.097451
Validation Loss decreased: (1.097480 --> 1.097451). Saving..
Epoch: 6        Training_Loss: 1.097326         Validation_Loss: 1.097403
Validation Loss decreased: (1.097451 --> 1.097403). Saving..
Epoch: 7        Training_Loss: 1.097270         Validation_Loss: 1.097375
Validation Loss decreased: (1.097403 --> 1.097375). Saving..
Epoch: 8        Training_Loss: 1.097451         Validation_Loss: 1.097343
Validation Loss decreased: (1.097375 --> 1.097343). Saving..
Epoch: 9        Training_Loss: 1.097602         Validation_Loss: 1.097321
Validation Loss decreased: (1.097343 --> 1.097321). Saving..
Epoch: 10       Training_Loss: 1.097193         Validation_Loss: 1.097301
Validation Loss decreased: (1.097321 --> 1.097301). Saving..
Epoch: 11       Training_Loss: 1.097169         Validation_Loss: 1.097291
Validation Loss decreased: (1.097301 --> 1.097291). Saving..
Epoch: 12       Training_Loss: 1.097084         Validation_Loss: 1.097290
Validation Loss decreased: (1.097291 --> 1.097290). Saving..
Epoch: 13       Training_Loss: 1.096981         Validation_Loss: 1.097275
Validation Loss decreased: (1.097290 --> 1.097275). Saving..
Epoch: 14       Training_Loss: 1.097114         Validation_Loss: 1.097262
Validation Loss decreased: (1.097275 --> 1.097262). Saving..
Epoch: 15       Training_Loss: 1.097103         Validation_Loss: 1.097247
Validation Loss decreased: (1.097262 --> 1.097247). Saving..
Epoch: 16       Training_Loss: 1.096397         Validation_Loss: 1.097230
Validation Loss decreased: (1.097247 --> 1.097230). Saving..
Epoch: 17       Training_Loss: 1.097071         Validation_Loss: 1.097230
Validation Loss decreased: (1.097230 --> 1.097230). Saving..
Epoch: 18       Training_Loss: 1.096804         Validation_Loss: 1.097215
Validation Loss decreased: (1.097230 --> 1.097215). Saving..
Epoch: 19       Training_Loss: 1.097510         Validation_Loss: 1.097197
Validation Loss decreased: (1.097215 --> 1.097197). Saving..
Epoch: 20       Training_Loss: 1.096473         Validation_Loss: 1.097160
Validation Loss decreased: (1.097197 --> 1.097160). Saving..
Epoch: 21       Training_Loss: 1.096781         Validation_Loss: 1.097154
Validation Loss decreased: (1.097160 --> 1.097154). Saving..
Epoch: 22       Training_Loss: 1.096599         Validation_Loss: 1.097136
Validation Loss decreased: (1.097154 --> 1.097136). Saving..
Epoch: 23       Training_Loss: 1.096814         Validation_Loss: 1.097105
Validation Loss decreased: (1.097136 --> 1.097105). Saving..
Epoch: 24       Training_Loss: 1.096466         Validation_Loss: 1.097070
Validation Loss decreased: (1.097105 --> 1.097070). Saving..
Training process is now completed
```

We load the best classification model

In [71]:
```
#load the best trained model
model.load_state_dict(torch.load(r'D:\dong;s junior (WSU)\Second Year - 2023\Semester 2\Machine Learning\Assignment\Assignment 2 - due 12 Nov\bestCN
```

Out[71]: <All keys matched successfully>

Calculate the accuracy score of the model

In [72]:
```
#Test the trained model
for i, data in enumerate(test_loader, 0):
    inputs, targets = data
    acc, y = Accuracy_score(model(inputs),targets)
    if i == 0:
        pred_y = y
        true_y = targets
    else:
        pred_y = torch.cat((pred_y, y))
        true_y = torch.cat((true_y, targets))

acc,_ = Accuracy_score(pred_y,true_y)
print("Total accuracy: ",acc.detach().numpy())
```

```
Total accuracy:  0.39
```

Overall, the total accuracy of the supervised model above is around 39%, which means this model is fairly acceptable as its total accuracy score is not high enough to accurately classify the images' labels. Further improvement could have been done to increase model's accuracy by increasing the number of epochs trained, adding more convolutional layers, etc.

The End