Full Name: Javad Razi

Student ID: 401204354

Introduction

In this practical assignment, we will practice using PyTorch to implement neural networks. We will work with the Fashion-MNIST dataset, and after visualizing the data to get familiar with it, we will use t-SNE and implement the PCA algorithm to reduce the dimensionality of the data for visualization in a 2D plot.

Tasks

1. Implement the PCA algorithm.
2. Apply PCA and t-SNE to the Fashion-MNIST test set.
3. Train a simple stacked autoencoder consisting of several MLP layers.
4. Use PCA and t-SNE to visualize the encoding of the test set calculated by the trained autoencoder.
5. Add a classification layer on top of the autoencoder's encoder and use its trained weights to predict each image label.

Instructions

1. For task 1, you can use the following steps:

   a. Import the necessary libraries.

   b. Define the PCA class.

   c. Instantiate the PCA class and fit it to the Fashion-MNIST test set.

   d. Transform the Fashion-MNIST test set using the fitted PCA model.

2. For task 2, you can use the following steps:

   a. Import the necessary libraries.

   b. Instantiate the PCA and t-SNE classes.

   c. Transform the Fashion-MNIST test set using the PCA model.

   d. Transform the PCA-transformed Fashion-MNIST test set using the t-SNE model.

   e. Visualize the t-SNE-transformed Fashion-MNIST test set using a scatter plot.

3. For task 3, you can use the following steps:

   a. Import the necessary libraries.

   b. Define the stacked autoencoder class.

c. Instantiate the stacked autoencoder class and train it on the Fashion-MNIST training set.

4. For task 4, you can use the following steps:

   a. Transform the Fashion-MNIST test set using the PCA model.

   b. Transform the PCA-transformed Fashion-MNIST test set using the trained autoencoder's encoder.

   c. Visualize the t-SNE-transformed encoding of the Fashion-MNIST test set using a scatter plot.

5. For task 5, you can use the following steps:

   a. Import the necessary libraries.

   b. Define the classification layer class.

   c. Add the classification layer to the trained autoencoder's encoder.

   d. Train the classification layer on the Fashion-MNIST training set.

   e. Evaluate the classification layer on the Fashion-MNIST test set.

Conclusion

In this practical assignment, we have learned how to use PyTorch to implement neural networks and apply dimensionality reduction techniques to data visualization. We have also learned how to train and evaluate classification models.

Import Requirements

```
import numpy as np
import random
import torch
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_openml
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from random import sample
```

Config

```
RANDOM_STATE = 42
random.seed(RANDOM_STATE)
```

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(device)
```

```
cuda
```

Load Fashion MNIST Dataset

```
fashion_mnist = fetch_openml(name='Fashion-MNIST')

X, y = fashion_mnist.data.astype('float32'), fashion_mnist.target.astype('int')
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/datasets/_openml.py:968: FutureWarning: The
  warn(
```

```
class_names = [
    "T-shirt/top", "Trouser", "Pullover", "Dress",
    "Coat", "Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot"
]

print(X.shape)
```

```
(70000, 784)
```

```
print(y.shape)
```

```
(70000,)
```

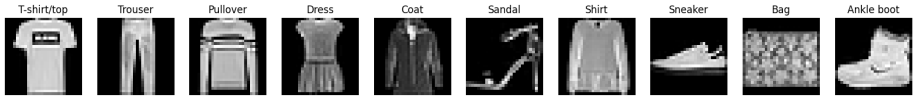
```
import numpy as np
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X.to_numpy(), y.to_numpy(), test_size=0.
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.075, random_

print(f'Shape of Training Data: {X_train.shape}')
print(f'Shape of Test Data: {X_test.shape}')
print(f'Shape of Validation Data: {X_val.shape}')
```

```
Shape of Training Data: (58275, 784)
Shape of Test Data: (7000, 784)
Shape of Validation Data: (4725, 784)
```

Visualization

```
def visualize_one_image_per_category(X, y, class_names):
    unique_labels = np.unique(y)
    fig, ax = plt.subplots(1, len(unique_labels), figsize=(20, 4))

    for i, label in enumerate(unique_labels):
        # Find the first image for the current label
        image_idx = np.where(y == label)[0][0]
        image = X.iloc[image_idx].values.reshape(28, 28)  # Use .iloc to access by index

        ax[i].imshow(image, cmap='gray')
        ax[i].set_title(class_names[label])
```

```
        ax[i].axis('off')

    plt.show()

visualize_one_image_per_category(X, y, class_names)
```



Dimensionality Reduction

```
def visualize_2d(reduced_data, labels, method='PCA', class_names=class_names):
    plt.figure(figsize=(12, 6))
    scatter = plt.scatter(reduced_data[:, 0], reduced_data[:, 1], c=labels, cmap='tab10', al
    plt.title(f'{method} Visualization')
    plt.xlabel(f'{method} Component 1')
    plt.ylabel(f'{method} Component 2')

    plt.colorbar(scatter, ticks=np.arange(len(class_names)), label='Class',
                 format=plt.FuncFormatter(lambda val, loc: class_names[int(val)]))

    for i, label in enumerate(class_names):
        indices = labels == i
        plt.text(reduced_data[indices, 0].mean(), reduced_data[indices, 1].mean(),
                 label, fontsize=10, ha='center', va='center')

    plt.show()
```

For this part and the following part, apply PCA and t-SNE to test dataset, respectively.

PCA (10 points)

Principal Component Analysis (PCA) - Step-by-Step Explanation

Principal Component Analysis (PCA) is a technique used for dimensionality reduction and data visualization. It aims to transform a dataset into a new coordinate system (a lower-dimensional space) while retaining the most important information.

1. **Center the Data**:
   - Calculate the mean of each feature in the dataset.
   - Subtract the mean from each feature value to center the data around the origin. This ensures that the new coordinate system is centered at the origin.

2. **Calculate Covariance Matrix**:
   - Compute the covariance matrix for the centered data.
   - The covariance matrix provides information about how features vary with each other.
3. **Calculate Eigenvectors and Eigenvalues**:
   - Compute the eigenvectors and eigenvalues of the covariance matrix.
   - Eigenvectors represent the directions of maximum variance, and eigenvalues quantify the amount of variance in those directions.
4. **Sort Eigenvectors**:
   - Sort the eigenvectors based on their corresponding eigenvalues in descending order.
   - The eigenvectors with higher eigenvalues capture more variance and are prioritized.
5. **Select Principal Components**:
   - Choose the top `n` eigenvectors (principal components) based on the desired number of dimensions for the reduced dataset.
   - These eigenvectors represent the directions in the original feature space that capture the most variance.
6. **Project Data**:
   - Project the centered data onto the lower-dimensional subspace formed by the selected principal components.
   - Multiply the centered data by the selected eigenvectors to obtain the new representation in the lower-dimensional space.
7. **Transform the Original Data:**:
   - Multiply the original data by the projection matrix to obtain the new lower-dimensional representation of the data.

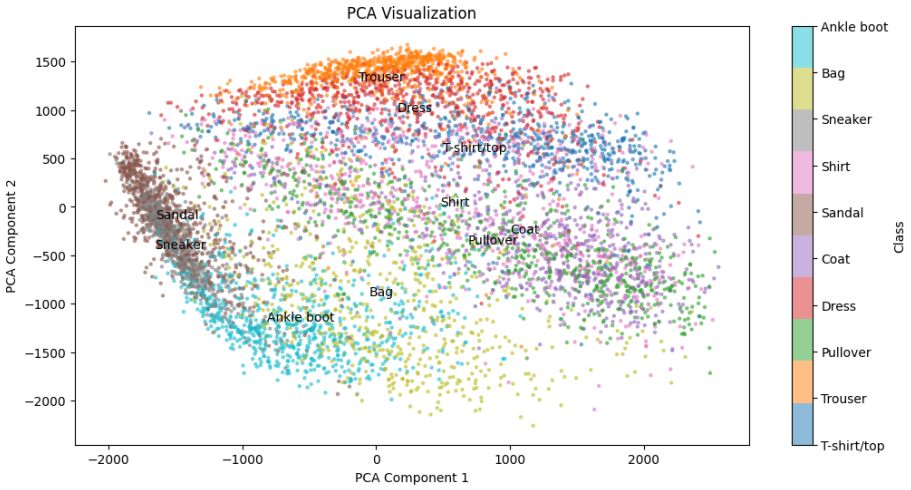
PCA helps in reducing the dimensionality of the dataset while retaining the most critical information. The first few principal components capture the majority of the variance, allowing for effective visualization and analysis of the data in a lower-dimensional space.

```
import numpy as np

class PCA:
    def __init__(self, n_components):
        self.n_components = n_components
        self.means = None
        self.eigenvectors = None

    def fit(self, X):
        self.means = np.mean(X, axis=0)
        X_centered = X - self.means

        covariance_matrix = np.cov(X_centered, rowvar=False)

        eigenvalues, eigenvectors = np.linalg.eigh(covariance_matrix)
```

```
        idx = np.argsort(eigenvalues)[::-1]
        eigenvalues = eigenvalues[idx]
        eigenvectors = eigenvectors[:, idx]

        self.eigenvectors = eigenvectors[:, :self.n_components]

    def transform(self, X):
        X_centered = X - self.means

        return np.dot(X_centered, self.eigenvectors)

    def fit_transform(self, X):
        self.fit(X)
        return self.transform(X)
```

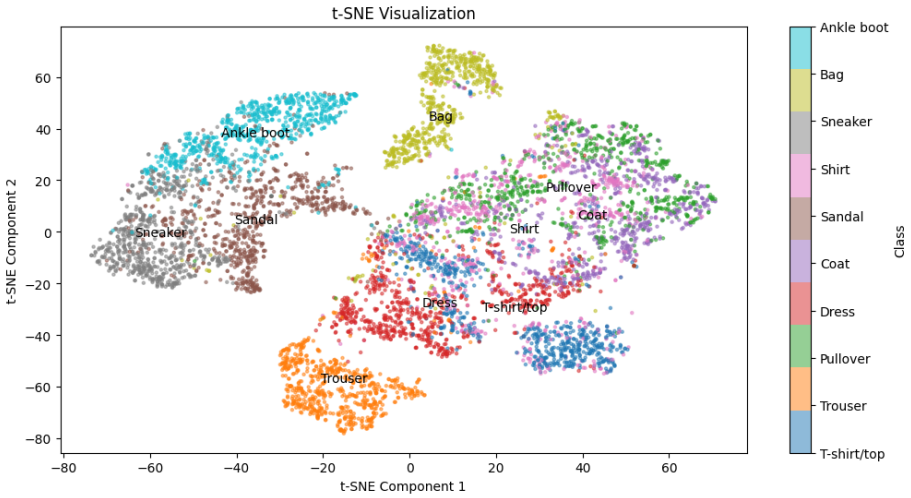
Apply PCA and t-SNE to the testset.

```
pca = PCA(2)
X_pca = pca.fit_transform(X_test)
visualize_2d(X_pca, y_test, 'PCA')
```



t-SNE

```
tsne = TSNE(n_components=2, random_state=RANDOM_STATE)
X_tsne = tsne.fit_transform(X_test)
visualize_2d(X_tsne, y_test, 't-SNE')
```



Question: (10 points)

- Explain the differences between PCA (Principal Component Analysis) and t-SNE (t-distributed Stochastic Neighbor Embedding) in terms of their preservation of distance, handling of non-linearity, and preservation of data structure. How does each technique aim to maintain distance or similarity relationships between data points in the lower-dimensional space, handle non-linear relationships in the data, and preserve the global or local data structure? Provide examples or illustrations to support your explanation.

Solution:

PCA and t-SNE are both techniques used for dimensionality reduction, but they differ significantly in their approaches and outcomes:

1. Preservation of Distance: PCA preserves linear variance and projects data into a new coordinate system where the greatest variances lie on the axes. It maintains global structure but can fail to capture complex, non-linear relationships. t-SNE, on the other hand, preserves local distances and similarities, especially the structure of the nearest neighbors. It can capture non-linear structures by converting high-dimensional Euclidean distances into conditional probabilities representing similarities.

2. Handling of Non-linearity: PCA is a linear technique and is not designed to reduce dimensions based on non-linear relationships. t-SNE excels in handling non-linear data structures. It maps complex manifolds by focusing on local relationships, often revealing clusters at several scales on a single map.

3. Preservation of Data Structure: PCA tends to preserve the global data structure and can sometimes blend distinct, local clusters if the global variance is more pronounced in other areas of the data. t-SNE preserves local data structures and can separate clusters that are not distinguishable by PCA. However, t-SNE does not preserve global relationships and distances between clusters can be arbitrary.

Both PCA and t-SNE aim to maintain certain relationships between data points when mapping to a lower-dimensional space. PCA maintains the global structure by projecting the data along axes of maximum variance, while t-SNE maintains local similarities and is more focused on revealing local clusters and data patterns.

Examples: In an image dataset where global brightness varies more than the content of the images, PCA might prioritize brightness variance over the actual image content, while t-SNE would group similar images regardless of brightness. In a dataset with nested clusters, PCA might project the data in a way that the clusters overlap, while t-SNE would unfold the manifolds and separate the clusters in the lower-dimensional space.

Deep Auto-Encoder For Representation Learning

In the previous section, you observed the results of applying two renowned dimensionality reduction techniques to the data. Additionally, a representation of the data can be learned by an **autoencoder**, which is a neural network that takes an image (or a noisy version of it) as input and attempts to reconstruct the image after encoding the pixels.

Create Dataset & Datalodaer (5 points)

- Note: If you are unfamiliar with PyTorch's `Dataset`, `Transforms`, and `Dataloader` modules, consult the following link for assistance: https://pytorch.org/tutorials/beginner/basics/data_tutorial.html

```
import torch
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms
from PIL import Image

class CustomFashionMNISTDataset(Dataset):
    def __init__(self, X, y, transform=None):
        self.X = X
        self.y = y
        self.transform = transform

    def __len__(self):
        return len(self.X)
```

```
    def __getitem__(self, idx):
        image = self.X[idx].reshape(28, 28).astype('uint8')
        label = self.y[idx]

        image = Image.fromarray(image)

        if self.transform:
            image = self.transform(image)

        return image, label

BATCH_SIZE = 64
train_transform = transforms.Compose([transforms.ToTensor()])
test_transform = transforms.Compose([transforms.ToTensor()])

train_dataset = CustomFashionMNISTDataset(X_train, y_train, train_transform)
val_dataset = CustomFashionMNISTDataset(X_val, y_val, test_transform)
test_dataset = CustomFashionMNISTDataset(X_test, y_test, test_transform)

train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=BATCH_SIZE, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE, shuffle=False)
```

Define Model (5 points)

Caution: You may only use multilayer perceptron (MLP) layers.

- Note: If you are unfamiliar with custom models using PyTorch, consult the following link for assistance: https://pytorch.org/tutorials/beginner/examples_nn/polynomial_module.html

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

class Autoencoder(nn.Module):
    def __init__(self):
        super(Autoencoder, self).__init__()
        # Encoder layers
        self.encoder = nn.Sequential(
            nn.Linear(28*28, 128),
            nn.ReLU(),
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, 12),
            nn.ReLU(),
```

```
            nn.Linear(12, 3),  # Compressed representation
        )
        # Decoder layers
        self.decoder = nn.Sequential(
            nn.Linear(3, 12),
            nn.ReLU(),
            nn.Linear(12, 64),
            nn.ReLU(),
            nn.Linear(64, 128),
            nn.ReLU(),
            nn.Linear(128, 28*28),
            nn.Sigmoid(),  # Output a value between 0 and 1
        )

    def forward(self, x):
        x = x.view(x.size(0), -1)  # Flatten the input
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
        return decoded

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
autoencoder = Autoencoder()
autoencoder.to(device)
```

```
Autoencoder(
  (encoder): Sequential(
    (0): Linear(in_features=784, out_features=128, bias=True)
    (1): ReLU()
    (2): Linear(in_features=128, out_features=64, bias=True)
    (3): ReLU()
    (4): Linear(in_features=64, out_features=12, bias=True)
    (5): ReLU()
    (6): Linear(in_features=12, out_features=3, bias=True)
  )
  (decoder): Sequential(
    (0): Linear(in_features=3, out_features=12, bias=True)
    (1): ReLU()
    (2): Linear(in_features=12, out_features=64, bias=True)
    (3): ReLU()
    (4): Linear(in_features=64, out_features=128, bias=True)
    (5): ReLU()
    (6): Linear(in_features=128, out_features=784, bias=True)
    (7): Sigmoid()
  )
)
```

Trainin Loop (10 points)

Complete the `train_at_epoch`, `test_ae`, and `train_ae` functions.

Define your `Optimizer`, `Learning Rate Scheduler`, and `Criterion` inside the `train_ae` function.

```
from torch import optim
import torch.nn as nn

def train_ae_epoch(model, train_loader, criterion, optimizer, device):
    model.train()
    total_loss = 0
    for batch_idx, (data, _) in enumerate(train_loader):
        data = data.view(data.size(0), -1).to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output, data)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    mean_loss = total_loss / len(train_loader)
    return mean_loss

def test_ae(model, test_loader, criterion, device):
    model.eval()
    total_loss = 0
    with torch.no_grad():
        for data, _ in test_loader:
            data = data.view(data.size(0), -1).to(device)
            output = model(data)
            loss = criterion(output, data)
            total_loss += loss.item()
    mean_loss = total_loss / len(test_loader)
    return mean_loss

def train_ae(model, train_loader, val_loader, num_epochs, learning_rate=1e-4, device=device)
    criterion = nn.MSELoss()
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)
    train_loss_history = []
    val_loss_history = []
    for epoch in range(num_epochs):
        train_loss = train_ae_epoch(model, train_loader, criterion, optimizer, device)
        val_loss = test_ae(model, val_loader, criterion, device)
        train_loss_history.append(train_loss)
        val_loss_history.append(val_loss)
```

```
        print(f'Epoch {epoch+1}/{num_epochs}, Train Loss: {train_loss:.4f}, Val Loss: {val_l
    return train_loss_history, val_loss_history

ae_train_loss_history, ae_val_loss_history = train_ae(autoencoder, train_loader, val_loader,
```

```
Epoch 1/50, Train Loss: 0.0883, Val Loss: 0.0572
Epoch 2/50, Train Loss: 0.0457, Val Loss: 0.0398
Epoch 3/50, Train Loss: 0.0380, Val Loss: 0.0368
Epoch 4/50, Train Loss: 0.0347, Val Loss: 0.0333
Epoch 5/50, Train Loss: 0.0324, Val Loss: 0.0319
Epoch 6/50, Train Loss: 0.0312, Val Loss: 0.0309
Epoch 7/50, Train Loss: 0.0303, Val Loss: 0.0300
Epoch 8/50, Train Loss: 0.0296, Val Loss: 0.0294
Epoch 9/50, Train Loss: 0.0290, Val Loss: 0.0289
Epoch 10/50, Train Loss: 0.0286, Val Loss: 0.0285
Epoch 11/50, Train Loss: 0.0282, Val Loss: 0.0282
Epoch 12/50, Train Loss: 0.0279, Val Loss: 0.0279
Epoch 13/50, Train Loss: 0.0276, Val Loss: 0.0276
Epoch 14/50, Train Loss: 0.0274, Val Loss: 0.0274
Epoch 15/50, Train Loss: 0.0271, Val Loss: 0.0271
Epoch 16/50, Train Loss: 0.0269, Val Loss: 0.0269
Epoch 17/50, Train Loss: 0.0267, Val Loss: 0.0267
Epoch 18/50, Train Loss: 0.0266, Val Loss: 0.0266
Epoch 19/50, Train Loss: 0.0264, Val Loss: 0.0265
Epoch 20/50, Train Loss: 0.0263, Val Loss: 0.0263
Epoch 21/50, Train Loss: 0.0261, Val Loss: 0.0262
Epoch 22/50, Train Loss: 0.0260, Val Loss: 0.0261
Epoch 23/50, Train Loss: 0.0259, Val Loss: 0.0260
Epoch 24/50, Train Loss: 0.0258, Val Loss: 0.0259
Epoch 25/50, Train Loss: 0.0257, Val Loss: 0.0258
Epoch 26/50, Train Loss: 0.0257, Val Loss: 0.0258
Epoch 27/50, Train Loss: 0.0256, Val Loss: 0.0257
Epoch 28/50, Train Loss: 0.0255, Val Loss: 0.0256
Epoch 29/50, Train Loss: 0.0254, Val Loss: 0.0255
Epoch 30/50, Train Loss: 0.0254, Val Loss: 0.0255
Epoch 31/50, Train Loss: 0.0253, Val Loss: 0.0253
Epoch 32/50, Train Loss: 0.0252, Val Loss: 0.0254
Epoch 33/50, Train Loss: 0.0252, Val Loss: 0.0252
Epoch 34/50, Train Loss: 0.0251, Val Loss: 0.0251
Epoch 35/50, Train Loss: 0.0250, Val Loss: 0.0251
Epoch 36/50, Train Loss: 0.0250, Val Loss: 0.0250
Epoch 37/50, Train Loss: 0.0249, Val Loss: 0.0249
Epoch 38/50, Train Loss: 0.0248, Val Loss: 0.0249
Epoch 39/50, Train Loss: 0.0248, Val Loss: 0.0248
Epoch 40/50, Train Loss: 0.0247, Val Loss: 0.0247
Epoch 41/50, Train Loss: 0.0247, Val Loss: 0.0247
Epoch 42/50, Train Loss: 0.0246, Val Loss: 0.0247
```

```
Epoch 43/50, Train Loss: 0.0245, Val Loss: 0.0246
Epoch 44/50, Train Loss: 0.0245, Val Loss: 0.0246
Epoch 45/50, Train Loss: 0.0244, Val Loss: 0.0245
Epoch 46/50, Train Loss: 0.0244, Val Loss: 0.0244
Epoch 47/50, Train Loss: 0.0243, Val Loss: 0.0244
Epoch 48/50, Train Loss: 0.0243, Val Loss: 0.0243
Epoch 49/50, Train Loss: 0.0242, Val Loss: 0.0243
Epoch 50/50, Train Loss: 0.0242, Val Loss: 0.0243
```

Advantages of Using PyTorch Lightning for Training Neural Networks

When working on training neural networks, it's essential to choose the right tools and frameworks to ensure efficient development, robustness, and maintainability of the code. PyTorch Lightning is a popular and powerful framework that simplifies the training process and offers several advantages over writing the training loop from scratch. In this exercise, we will explore these benefits by comparing manual implementation with PyTorch Lightning.

1. **Structured and Readable Code:** PyTorch Lightning enforces a clear structure by separating the PyTorch components (such as model, optimizer, and scheduler) into dedicated methods like training_step, validation_step, and configure_optimizers. This separation results in more readable and organized code, making it easier to understand and maintain.

2. **Reduced Boilerplate Code:** Writing a training loop involves a significant amount of boilerplate code for handling various aspects of training, such as iterating over the dataset, updating parameters, and logging metrics. PyTorch Lightning abstracts away much of this boilerplate, allowing you to focus on the essential components of your model and experiment.

3. **Flexibility and Customization:** Despite providing a high-level interface, PyTorch Lightning remains flexible and allows for customization. Users can override specific methods to tailor the training process to their needs while leveraging the standardized structure provided by the framework.

4. **Enhanced Reproducibility:** PyTorch Lightning promotes code modularity and follows best practices, contributing to enhanced reproducibility. With a consistent structure across experiments, it becomes easier to replicate and compare results.

5. **Integration with Advanced Features:** PyTorch Lightning seamlessly integrates with advanced features such as distributed training, mixed-precision training, and automatic optimization, among others. These features are often complex to implement manually but can be easily utilized with PyTorch Lightning.

In summary, PyTorch Lightning provides a high-level and well-structured interface for training neural networks, offering benefits such as code readability,

reduced boilerplate, flexibility, reproducibility, integration with advanced features, and a vibrant community. By using PyTorch Lightning, we can expedite the development process, enhance code quality, and facilitate experimentation and research in the field of deep learning.**

Migrate to PL

In addition to all components defined in the preceding training phase, incorporate the early stopping module from the PyTorch Lightning API and a model checkpoint that saves the best model in each epoch.

```
%pip install pytorch-lightning
```

```
Collecting pytorch-lightning
  Downloading pytorch_lightning-2.1.1-py3-none-any.whl (776 kB)
                     776.3/776.3 kB 9.7 MB/s eta 0:00:00
ent already satisfied: numpy>=1.17.2 in /usr/local/lib/python3.10/dist-packages (from pytorc
Requirement already satisfied: torch>=1.12.0 in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: tqdm>=4.57.0 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: PyYAML>=5.4 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: fsspec[http]>2021.06.0 in /usr/local/lib/python3.10/dist-pack
Collecting torchmetrics>=0.7.0 (from pytorch-lightning)
  Downloading torchmetrics-1.2.0-py3-none-any.whl (805 kB)
                     805.2/805.2 kB 17.6 MB/s eta 0:00:00
ent already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from pyto
Requirement already satisfied: typing-extensions>=4.0.0 in /usr/local/lib/python3.10/dist-pa
Collecting lightning-utilities>=0.8.0 (from pytorch-lightning)
  Downloading lightning_utilities-0.9.0-py3-none-any.whl (23 kB)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from fss
Requirement already satisfied: aiohttp!=4.0.0a0,!=4.0.0a1 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from tor
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch>
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from tor
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch
Requirement already satisfied: triton==2.1.0 in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: charset-normalizer<4.0,>=2.0 in /usr/local/lib/python3.10/dis
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: async-timeout<5.0,>=4.0.0a3 in /usr/local/lib/python3.10/dist
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from
```

```
Installing collected packages: lightning-utilities, torchmetrics, pytorch-lightning
Successfully installed lightning-utilities-0.9.0 pytorch-lightning-2.1.1 torchmetrics-1.2.0
```

Complete The Code For the PL Trainer (15 point)

```python
import torch
import torch.nn as nn
import torch.optim as optim
import pytorch_lightning as pl
from pytorch_lightning.callbacks import EarlyStopping, ModelCheckpoint

class AE_Trainer(pl.LightningModule):
    def __init__(self, model):
        super(AE_Trainer, self).__init__()
        self.model = model
        self.criterion = nn.MSELoss()
        self.train_losses = []  # To store training loss history
        self.val_losses = []    # To store validation loss history

    def forward(self, x):
        return self.model(x)

    def training_step(self, batch, batch_idx):
        x, _ = batch
        x = x.view(x.size(0), -1)
        x_hat = self.model(x)
        loss = self.criterion(x_hat, x)
        self.log('train_loss', loss, on_step=True, on_epoch=True, prog_bar=True, logger=True
        return loss

    def on_train_epoch_end(self):
        train_loss_avg = self.trainer.callback_metrics['train_loss_epoch']
        self.train_losses.append(train_loss_avg.item())

    def validation_step(self, batch, batch_idx):
        x, _ = batch
        x = x.view(x.size(0), -1)
        x_hat = self.model(x)
        loss = self.criterion(x_hat, x)
        self.log('val_loss', loss, on_step=False, on_epoch=True, prog_bar=True, logger=True)
        return loss

    def on_validation_epoch_end(self):
        val_loss_avg = self.trainer.callback_metrics['val_loss']
        self.val_losses.append(val_loss_avg.item())
```

```
    def configure_optimizers(self):
        optimizer = torch.optim.Adam(self.model.parameters(), lr=1e-3)
        scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=1, gamma=0.9)
        return {
            'optimizer': optimizer,
            'lr_scheduler': {
                'scheduler': scheduler,
                'monitor': 'val_loss',
            },
        }


autoencoder = Autoencoder()
ae_trainer = AE_Trainer(autoencoder)

early_stopping = EarlyStopping('val_loss', patience=5)
checkpoint_callback = ModelCheckpoint(dirpath='checkpoints/', save_top_k=1, monitor='val_los

trainer = pl.Trainer(
    max_epochs=50,
    accelerator='gpu' if torch.cuda.is_available() else 'cpu',
    callbacks=[early_stopping, checkpoint_callback],
)

trainer.fit(ae_trainer, train_loader, val_loader)
autoencoder = ae_trainer.model
```

```
INFO:pytorch_lightning.utilities.rank_zero:GPU available: True (cuda), used: True
INFO:pytorch_lightning.utilities.rank_zero:TPU available: False, using: 0 TPU cores
INFO:pytorch_lightning.utilities.rank_zero:IPU available: False, using: 0 IPUs
INFO:pytorch_lightning.utilities.rank_zero:HPU available: False, using: 0 HPUs
/usr/local/lib/python3.10/dist-packages/pytorch_lightning/callbacks/model_checkpoint.py:634
INFO:pytorch_lightning.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
INFO:pytorch_lightning.callbacks.model_summary:
  | Name      | Type        | Params
-----------------------------------------
0 | model     | Autoencoder | 219 K
1 | criterion | MSELoss     | 0
-----------------------------------------
219 K     Trainable params
0         Non-trainable params
219 K     Total params
0.880     Total estimated model params size (MB)
```

{"model_id":"ee2713464bcf40dd9e9b9fd97d5fe545","version_major":2,"version_minor":0}

```
/usr/local/lib/python3.10/dist-packages/pytorch_lightning/trainer/connectors/data_connector
```

/usr/local/lib/python3.10/dist-packages/pytorch_lightning/trainer/connectors/data_connector.

{"model_id":"68cac88665804ddcb4e046455a22afc1","version_major":2,"version_minor":0}

{"model_id":"7d6d40733f2c4bf9905b63c2d5090f4a","version_major":2,"version_minor":0}

{"model_id":"4c94d117a2f146eba7896cffe35d81b0","version_major":2,"version_minor":0}

{"model_id":"a5a90d2fd7ff4404b84d7bd8f624ccd0","version_major":2,"version_minor":0}

{"model_id":"43549488c0a64b73a1f82b940409bfd7","version_major":2,"version_minor":0}

{"model_id":"8c4bb770710140f78205fb0157164cf0","version_major":2,"version_minor":0}

{"model_id":"cb3e33f0d54940f6bf3cdc7b16cf23e6","version_major":2,"version_minor":0}

{"model_id":"270b69571fa245e19730e16a00093389","version_major":2,"version_minor":0}

{"model_id":"bb486524f17f4bea926cbdf7f673d132","version_major":2,"version_minor":0}

{"model_id":"63c9bafecad6494a9a337ccc67b4f25b","version_major":2,"version_minor":0}

{"model_id":"1bc308ebbd3b477d8dc0854515e5f259","version_major":2,"version_minor":0}

{"model_id":"021c069fab11405e9f288cce48f90027","version_major":2,"version_minor":0}

{"model_id":"95ba576373ed41f0acdc1b357d5c68ae","version_major":2,"version_minor":0}

{"model_id":"2ca105ce9e91419da6cdf1a164a1b7ac","version_major":2,"version_minor":0}

{"model_id":"2d6b5ac0f13049f79592cfa04a1c022d","version_major":2,"version_minor":0}

{"model_id":"1f76469165ed4d51bc29a10ffc5bc697","version_major":2,"version_minor":0}

{"model_id":"b4616c21c8c74c379071cdda0d82664a","version_major":2,"version_minor":0}

{"model_id":"a06ebdd3dfd04d8389b570af2609721a","version_major":2,"version_minor":0}

{"model_id":"b3a7510db23e43debfe9c142826ee3bc","version_major":2,"version_minor":0}

{"model_id":"1406ab037fca483cb16e62bc4aa2bc33","version_major":2,"version_minor":0}

{"model_id":"07f2694018394ad48d45485a8ec23dce","version_major":2,"version_minor":0}

{"model_id":"2d0a004149dc4fba9f91abe9731b4c41","version_major":2,"version_minor":0}

{"model_id":"28d2a531bf5844739c75dd4353e13ef1","version_major":2,"version_minor":0}

{"model_id":"e2ff14dc74234831b67c790a5bb5773b","version_major":2,"version_minor":0}

{"model_id":"c62eacd96c7545549ef9b35faea8764c","version_major":2,"version_minor":0}

{"model_id":"628d51d2a8a94fe4aea67fce85d49eb1","version_major":2,"version_minor":0}

{"model_id":"4a811b42cbd24aeb9621f48640c87ae7","version_major":2,"version_minor":0}

{"model_id":"d63633e322574cdfb3d1ff5c490b538f","version_major":2,"version_minor":0}

{"model_id":"53d6d069c9ed47c5974d4f2082d6a208","version_major":2,"version_minor":0}

{"model_id":"fba82f0f9fb84238a7277b5f05a66b17","version_major":2,"version_minor":0}

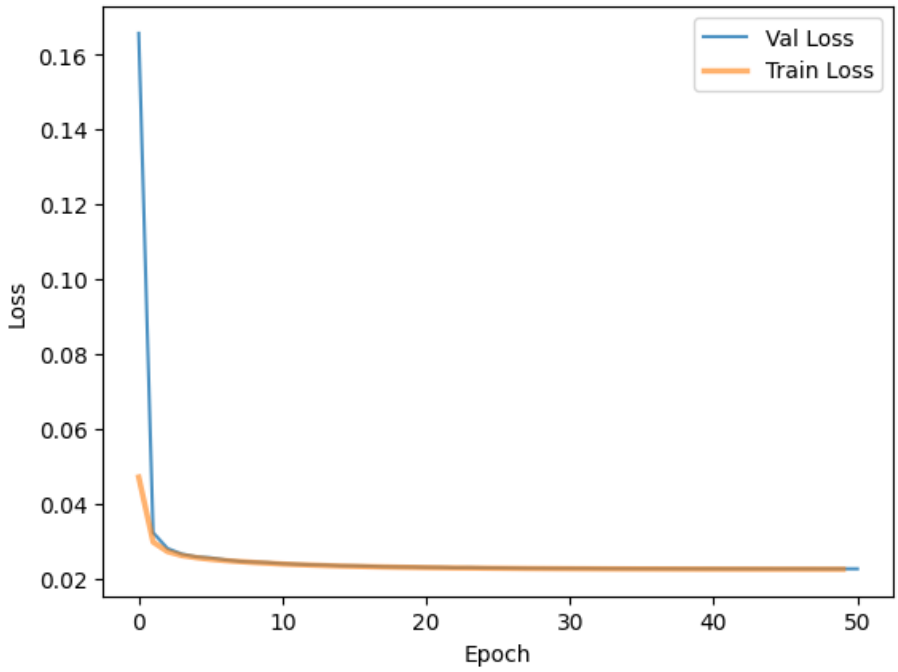
{"model_id":"0e270593e90b43e0a71164f184d380f3","version_major":2,"version_minor":0}
{"model_id":"2f358e3966cf48a1b63f80f2b1dc89ac","version_major":2,"version_minor":0}
{"model_id":"73e4797bafbf498b88fbf2e0312650aa","version_major":2,"version_minor":0}
{"model_id":"44e1491e886143d7a1052afb75415484","version_major":2,"version_minor":0}
{"model_id":"52ca7195eff64496ab92a07488839eb8","version_major":2,"version_minor":0}
{"model_id":"36e8dd27c27a4b089a2e7f8da7b8b0eb","version_major":2,"version_minor":0}
{"model_id":"8fcf601c08084d3ebb68e0c27388c3f6","version_major":2,"version_minor":0}
{"model_id":"1e2c00347ae44c0298a171971edb46ef","version_major":2,"version_minor":0}
{"model_id":"a918f09b15c14b35804e983f3b61b3b3","version_major":2,"version_minor":0}
{"model_id":"20cb096353564941ab7e62a127a84a12","version_major":2,"version_minor":0}
{"model_id":"4c3717947d574ebba1dcc1549cb41bdf","version_major":2,"version_minor":0}
{"model_id":"b8789280a98f4533a0c7357b518c4c6f","version_major":2,"version_minor":0}
{"model_id":"5b32ac6325314d40985444de8fce8026","version_major":2,"version_minor":0}
{"model_id":"5455a5edacc54fbca3312f6627f07bb9","version_major":2,"version_minor":0}
{"model_id":"088d81d3584141f8a12e2bec8c183488","version_major":2,"version_minor":0}
{"model_id":"01853dadc4924725b24fb06385fde37b","version_major":2,"version_minor":0}
{"model_id":"14142d2650d040138677b72fa6e5d01b","version_major":2,"version_minor":0}
{"model_id":"5bb5919fbfb34d09b35fd79e729cbbab","version_major":2,"version_minor":0}
{"model_id":"9fbeaf81f079421e87536fd7d62a9817","version_major":2,"version_minor":0}
{"model_id":"71554a68a7d84bffaf85122dedd0ac6f","version_major":2,"version_minor":0}
{"model_id":"5456233000a2433fa1890d471befd4b8","version_major":2,"version_minor":0}

```
INFO:pytorch_lightning.utilities.rank_zero:`Trainer.fit` stopped: `max_epochs=50` reached.
```

Visualize Losses

Visualize Train_Loss and Val_Loss durining the training phase.

```
plt.plot(ae_trainer.val_losses, label='Val Loss', linewidth=1.5, alpha=0.8)
plt.plot(ae_trainer.train_losses, label='Train Loss', linewidth=2.5, alpha=0.6)
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



Evaluate on Testset

```
if torch.cuda.is_available():
    device = 'cuda:0'

    autoencoder.to(device)

    test_loader = [(data.to(device), target) for data, target in test_loader]

test_ae(autoencoder, test_loader, nn.MSELoss(), device)
```

```
0.022352541170336984
```

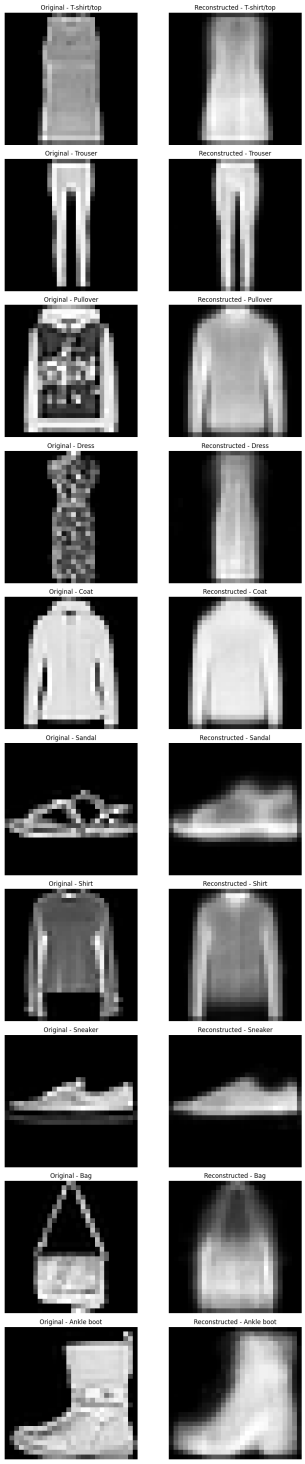
Visualize Model Output (8 points)

For each category in the test set:

1. Randomly select an image from that category.
2. Generate a reconstruction of the image.
3. Display the original image and its reconstruction side-by-side.

```
import matplotlib.pyplot as plt
from torchvision.utils import make_grid

def visualize_input_reconstructed(autoencoder, test_loader, device, class_names):
```

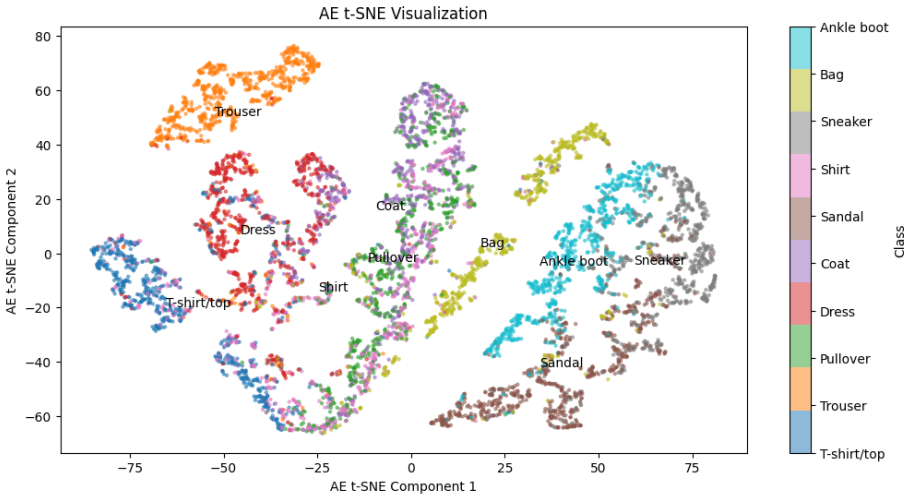
```
    autoencoder.eval()
    fig, axes = plt.subplots(nrows=len(class_names), ncols=2, figsize=(10, 4 * len(class_nam

    with torch.no_grad():
        for i, class_name in enumerate(class_names):
            # Find an image of the given class
            for images, labels in test_loader:
                idx = (labels == i).nonzero(as_tuple=True)[0][0]
                original_image = images[idx]
                break

            # Generate reconstruction
            reconstructed_image = autoencoder(original_image.view(1, -1).to(device)).cpu().v

            # Plot the original image
            axes[i, 0].imshow(original_image.cpu().squeeze(), cmap='gray')

            axes[i, 0].set_title(f'Original - {class_name}')
            axes[i, 0].axis('off')

            # Plot the reconstructed image
            axes[i, 1].imshow(reconstructed_image.cpu().squeeze(), cmap='gray')
            axes[i, 1].set_title(f'Reconstructed - {class_name}')
            axes[i, 1].axis('off')

    plt.tight_layout()
    plt.show()

visualize_input_reconstructed(autoencoder, test_loader, device, class_names)
```


Original - T-shirt/top
Reconstructed - T-shirt/top
Original - Trouser
Reconstructed - Trouser
Original - Pullover
Reconstructed - Pullover
Original - Dress
Reconstructed - Dress
Original - Coat
Reconstructed - Coat
Original - Sandal
Reconstructed - Sandal
Original - Shirt
Reconstructed - Shirt
Original - Sneaker
Reconstructed - Sneaker
Original - Bag
Reconstructed - Bag
Original - Ankle boot
Reconstructed - Ankle boot

Visualize Embeddings (5 points)

Generate image embeddings for the test dataset.

```
def generate_embeddings(autoencoder, test_loader, device):
    embeddings = []
    autoencoder.eval()
    with torch.no_grad():
        for images, _ in test_loader:
            images = images.to(device)
            encoded_images = autoencoder.encoder(images.view(images.size(0), -1))
            embeddings.append(encoded_images.cpu().numpy())
    embeddings = np.concatenate(embeddings, axis=0)
    return embeddings

embeddings = generate_embeddings(autoencoder, test_loader, device)

tsne = TSNE(n_components=2, random_state=RANDOM_STATE)
embeddings_tsne = tsne.fit_transform(embeddings)
visualize_2d(embeddings_tsne, y_test, 'AE t-SNE')
```



Train a Classifier (8 points)

In the previous part, we saw that the encoder has learned to project images into representations with semantic meaning. In this part, you will transfer the knowledge learned by the encoder to a classification task by adding classification layers on top of it. This is a common technique in transfer learning, and it can be used to improve the performance of a classifier on a new task, even if the classifier is trained on a limited amount of data

```
def get_encoder(autoencoder):
    encoder = autoencoder.encoder
    return nn.Sequential(*list(encoder.children())[:-1])

class Classifier(nn.Module):
    def __init__(self, encoder, num_classes):
        super(Classifier, self).__init__()
        self.encoder = encoder
        self.classifier_layer = nn.Sequential(
            nn.Linear(12, num_classes)
        )

    def forward(self, x):
        x = x.view(x.size(0), -1)
        x = self.encoder(x)
        x = self.classifier_layer(x)
        return x

encoder = get_encoder(autoencoder)
classifier = Classifier(encoder, len(class_names))
classifier.to(device)
```

```
Classifier(
  (encoder): Sequential(
    (0): Linear(in_features=784, out_features=128, bias=True)
    (1): ReLU()
    (2): Linear(in_features=128, out_features=64, bias=True)
    (3): ReLU()
    (4): Linear(in_features=64, out_features=12, bias=True)
    (5): ReLU()
  )
  (classifier_layer): Sequential(
    (0): Linear(in_features=12, out_features=10, bias=True)
  )
)
```

Training Loop (9 Points)

Implement the `train_clf_epoch`, `test_clf`, and `train_clf` functions. In the `test_clf` function, in addition to the loss, calculate and return the accuracy.

```
def train_clf_epoch(model, train_loader, criterion, optimizer, device):
    model.train()
    train_loss = 0.0
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()
```

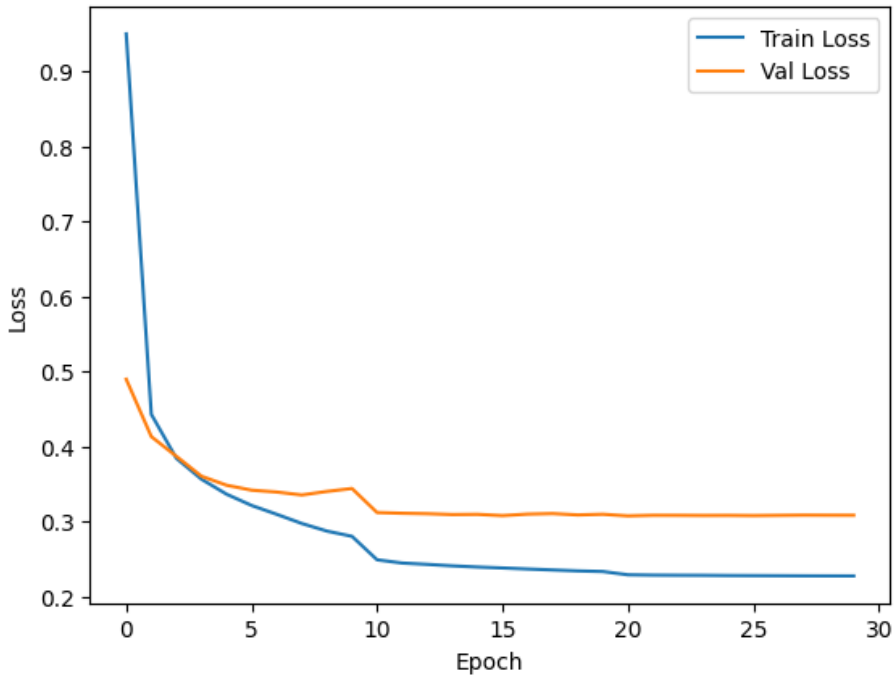
```
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            train_loss += loss.item()
        mean_loss = train_loss / len(train_loader)
        return mean_loss

    def test_clf(model, test_loader, criterion, device):
        model.eval()
        test_loss = 0.0
        correct = 0
        total = 0
        with torch.no_grad():
            for images, labels in test_loader:
                images, labels = images.to(device), labels.to(device)
                outputs = model(images)
                loss = criterion(outputs, labels)
                test_loss += loss.item()
                _, predicted = torch.max(outputs.data, 1)
                total += labels.size(0)
                correct += (predicted == labels).sum().item()
        accuracy = correct / total
        mean_loss = test_loss / len(test_loader)
        return mean_loss, accuracy

    def train_clf(model, train_loader, val_loader, num_epochs, device):
        learning_rate = 1e-3
        criterion = nn.CrossEntropyLoss()
        optimizer = optim.Adam(model.parameters(), lr=learning_rate)
        scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.1)
        train_loss_history = []
        val_loss_history = []
        val_acc_history = []

        for epoch in range(num_epochs):
            train_loss = train_clf_epoch(model, train_loader, criterion, optimizer, device)
            val_loss, val_acc = test_clf(model, val_loader, criterion, device)
            scheduler.step()

            print(f"Epoch {epoch+1}/{num_epochs}, Train Loss: {train_loss:.4f}, Val Loss: {val_l

            train_loss_history.append(train_loss)
            val_loss_history.append(val_loss)
            val_acc_history.append(val_acc)
```

```
    return train_loss_history, val_loss_history, val_acc_history

clf_train_loss_history, clf_val_loss_history, clf_val_acc_history = train_clf(classifier, tr
```

```
Epoch 1/30, Train Loss: 0.9498, Val Loss: 0.4893, Val ACC: 83.05
Epoch 2/30, Train Loss: 0.4426, Val Loss: 0.4129, Val ACC: 85.35
Epoch 3/30, Train Loss: 0.3839, Val Loss: 0.3866, Val ACC: 86.01
Epoch 4/30, Train Loss: 0.3561, Val Loss: 0.3601, Val ACC: 87.01
Epoch 5/30, Train Loss: 0.3362, Val Loss: 0.3481, Val ACC: 87.32
Epoch 6/30, Train Loss: 0.3211, Val Loss: 0.3415, Val ACC: 87.75
Epoch 7/30, Train Loss: 0.3093, Val Loss: 0.3390, Val ACC: 87.75
Epoch 8/30, Train Loss: 0.2971, Val Loss: 0.3351, Val ACC: 87.81
Epoch 9/30, Train Loss: 0.2870, Val Loss: 0.3398, Val ACC: 87.68
Epoch 10/30, Train Loss: 0.2800, Val Loss: 0.3437, Val ACC: 87.62
Epoch 11/30, Train Loss: 0.2487, Val Loss: 0.3116, Val ACC: 88.66
Epoch 12/30, Train Loss: 0.2444, Val Loss: 0.3107, Val ACC: 88.80
Epoch 13/30, Train Loss: 0.2426, Val Loss: 0.3101, Val ACC: 88.87
Epoch 14/30, Train Loss: 0.2406, Val Loss: 0.3090, Val ACC: 88.80
Epoch 15/30, Train Loss: 0.2391, Val Loss: 0.3093, Val ACC: 88.89
Epoch 16/30, Train Loss: 0.2378, Val Loss: 0.3076, Val ACC: 89.02
Epoch 17/30, Train Loss: 0.2364, Val Loss: 0.3096, Val ACC: 88.85
Epoch 18/30, Train Loss: 0.2352, Val Loss: 0.3103, Val ACC: 88.87
Epoch 19/30, Train Loss: 0.2339, Val Loss: 0.3085, Val ACC: 89.02
Epoch 20/30, Train Loss: 0.2331, Val Loss: 0.3094, Val ACC: 88.76
Epoch 21/30, Train Loss: 0.2287, Val Loss: 0.3072, Val ACC: 89.08
Epoch 22/30, Train Loss: 0.2283, Val Loss: 0.3080, Val ACC: 88.87
Epoch 23/30, Train Loss: 0.2281, Val Loss: 0.3080, Val ACC: 88.87
Epoch 24/30, Train Loss: 0.2280, Val Loss: 0.3078, Val ACC: 88.83
Epoch 25/30, Train Loss: 0.2277, Val Loss: 0.3079, Val ACC: 88.91
Epoch 26/30, Train Loss: 0.2276, Val Loss: 0.3076, Val ACC: 89.14
Epoch 27/30, Train Loss: 0.2275, Val Loss: 0.3079, Val ACC: 88.89
Epoch 28/30, Train Loss: 0.2273, Val Loss: 0.3083, Val ACC: 88.91
Epoch 29/30, Train Loss: 0.2272, Val Loss: 0.3081, Val ACC: 88.95
Epoch 30/30, Train Loss: 0.2272, Val Loss: 0.3081, Val ACC: 88.99
```

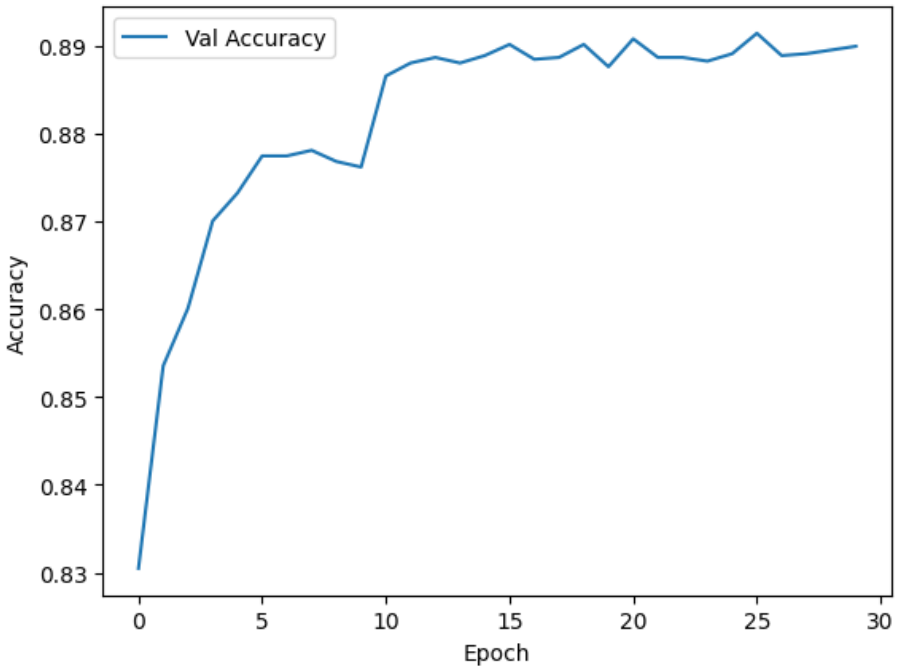
Visualize Losses

```
plt.plot(clf_train_loss_history, label='Train Loss')
plt.plot(clf_val_loss_history, label='Val Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



Visualize Validation Accuracy

```
plt.plot(clf_val_acc_history, label='Val Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



Evaluate on Testset (20 points)

Your model must achieve an accuracy of at least 90% on the test set to receive full marks.

```
test_loss, test_acc = test_clf(classifier, test_loader, nn.CrossEntropyLoss(), device)

print(f"Test Loss: {test_loss:.4f}, Test ACC: {100 * test_acc:.2f}")
```

```
Test Loss: 0.2904, Test ACC: 89.36
```