HW3 report

110511184 劉宇翔

1. Explain your implementation which got the best performance in detail.

Implementation Overview: For the anomaly detection task using the UCI Letter Image Recognition dataset, we used multiple approaches to achieve the best performance, which involved an ensemble of models. The final implementation included:

1. **Autoencoder for Anomaly Detection**: Architecture: A deep autoencoder with multiple layers and residual blocks to capture complex patterns in the data. Training: Used K-Fold cross-validation to ensure robustness and prevent overfitting. Each fold trained the autoencoder and validated its performance. Regularization Techniques: Implemented dropout and layer normalization to improve generalization and prevent overfitting. Early Stopping and Learning Rate Scheduling: Early stopping based on validation loss to avoid overfitting, and learning rate scheduler to adjust the learning rate dynamically.

2. **OneClass SVM**: Used the OneClass SVM algorithm with an RBF kernel to learn the decision boundary that separates the normal data from outliers. Scaled the features before feeding them into the model.

3. **K-Means Clustering**: Applied K-Means clustering to identify clusters within the data. Used the distance from each test point to the nearest cluster centroid as an anomaly score.

4. **Ensemble Method**: Combined the scores from the autoencoder, OneClass SVM, and K-Means to create a final ensemble score. Averaged the anomaly scores from the three methods to improve robustness and performance.

Detailed Code Explanation:

```
class Autoencoder(nn.Module):
    def __init__(self):
        super(Autoencoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(input_dim, 1024),
            nn.GELU(),
            nn.Dropout(0.3),
            nn.Linear(1024, 512),
            nn.GELU(),
            nn.Dropout(0.3),
            nn.Linear(512, 256),
            nn.GELU(),
            ResidualBlock(256),
            nn.Linear(256, 128),
            nn.GELU(),
            nn.Linear(128, 64),
            ResidualBlock(64)
        )
        self.decoder = nn.Sequential(
```

```
            nn.Linear(64, 128),
            nn.GELU(),
            nn.Dropout(0.3),
            nn.Linear(128, 256),
            nn.GELU(),
            nn.Dropout(0.3),
            nn.Linear(256, 512),
            nn.GELU(),
            ResidualBlock(512),
            nn.Linear(512, 1024),
            nn.GELU(),
            nn.Linear(1024, input_dim),
            ResidualBlock(input_dim)
        )

    def forward(self, x):
        x = self.encoder(x)
        x = self.decoder(x)
        return x
```

Training and Validation:

```
for fold, (train_idx, val_idx) in enumerate(kf.split(X_train)):
    train_loader = DataLoader(LetterDataset(X_train[train_idx]), batch_size=64,
shuffle=True)
    val_loader = DataLoader(LetterDataset(X_train[val_idx]), batch_size=64,
shuffle=False)

    model = Autoencoder().to(device)
    criterion = nn.MSELoss()
    optimizer = optim.Adam(model.parameters(), lr=0.001, weight_decay=1e-5)
    scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=10)

    best_loss = float('inf')
    early_stopping_counter = 0

    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0
        for X_batch in train_loader:
            X_batch = X_batch.to(device)
            optimizer.zero_grad()
            with torch.cuda.amp.autocast():
                outputs = model(X_batch)
                loss = criterion(outputs, X_batch)
            scaler.scale(loss).backward()
            scaler.step(optimizer)
            scaler.update()
            running_loss += loss.item()

        average_loss = running_loss / len(train_loader)
```

```
        model.eval()
        val_loss = 0.0
        with torch.no_grad():
            for X_batch in val_loader:
                X_batch = X_batch.to(device)
                outputs = model(X_batch)
                loss = criterion(outputs, X_batch)
                val_loss += loss.item()
        val_loss /= len(val_loader)

        scheduler.step()

        if val_loss < best_loss:
            best_loss = val_loss
            torch.save(model.state_dict(),
f'best_autoencoder_weights_fold{fold+1}.pth')
            early_stopping_counter = 0
        else:
            early_stopping_counter += 1
            if early_stopping_counter >= early_stopping_patience:
                break
```

OneClass SVM and K-Means:

```
oc_svm = OneClassSVM(kernel='rbf', gamma='scale')
oc_svm.fit(X_train)
oc_svm_scores = -oc_svm.decision_function(X_test)

kmeans = KMeans(n_clusters=6, random_state=42)
kmeans.fit(X_train)
kmeans_distances = kmeans.transform(X_test).min(axis=1)
```

Combining Results:

```
combined_scores = (reconstruction_errors + oc_svm_scores + kmeans_distances) / 3
submission_combined = pd.DataFrame({'id': range(len(combined_scores)), 'outliers':
combined_scores})
submission_combined.to_csv('submission_combined.csv', index=False)
```

2. Explain the rationale for using AUC score instead of F1 score for binary classification in this homework.

AUC (Area Under the Curve) Score

- **Definition**: AUC measures the area under the ROC (Receiver Operating Characteristic) curve, which plots the true positive rate (sensitivity) against the false positive rate (1-specificity) at various threshold

settings.
- **Interpretation**: AUC ranges from 0 to 1, where 1 represents a perfect model, 0.5 indicates a model with no discrimination ability (random guess), and less than 0.5 indicates a model performing worse than random guessing.

Rationale for Using AUC:

- T**hreshold Independence**: AUC evaluates the performance across all possible classification thresholds, providing a comprehensive measure of the model's ability to discriminate between classes. This is particularly useful in anomaly detection where choosing a specific threshold can be challenging.
- **Robustness to Class Imbalance**: AUC is less sensitive to class imbalance compared to metrics like F1 score. In anomaly detection, the number of outliers (anomalies) is typically much smaller than the number of normal instances, making AUC a more appropriate metric.
- **Focus on Ranking**: AUC evaluates the ranking of predictions rather than their absolute values. This aligns well with anomaly detection tasks where we are interested in ranking instances by their likelihood of being anomalies.

F1 Score:

- **Definition**: F1 score is the harmonic mean of precision and recall, providing a single measure of a test's accuracy.
- **Interpretation**: F1 score ranges from 0 to 1, where 1 indicates perfect precision and recall, and 0 indicates the worst performance.

Challenges with F1 Score in This Context:

- **Threshold Dependency**: F1 score requires a predefined classification threshold to calculate precision and recall. Selecting an appropriate threshold can be challenging, especially in an unsupervised anomaly detection context.
- **Class Imbalance Sensitivity**: F1 score can be significantly affected by class imbalance. In datasets with a small number of anomalies, a high precision might be achieved with a low recall, leading to a misleadingly high F1 score.

Given these considerations, AUC is preferred in this homework for its ability to provide a comprehensive and robust measure of the model's performance across different thresholds and in the presence of class imbalance.

3. Discuss the difference between semi-supervised learning and unsupervised learning.

Unsupervised Learning:

- **Definition**: Unsupervised learning involves training models on data without labeled responses. The goal is to identify patterns, structures, or relationships within the data.
- **Common Techniques**: Clustering (e.g., K-Means, DBSCAN), dimensionality reduction (e.g., PCA, t-SNE), and anomaly detection (e.g., OneClass SVM, autoencoders).
- **Applications**: Customer segmentation, anomaly detection, feature learning, and data compression.

Semi-Supervised Learning:

- **Definition**: Semi-supervised learning combines a small amount of labeled data with a large amount of unlabeled data during training. The goal is to leverage the unlabeled data to improve the learning accuracy and generalization.
- **Approaches**:
    - **Self-Training**: The model is initially trained on the labeled data, then used to predict labels for the unlabeled data. These predicted labels are used to retrain the model iteratively.
    - **Co-Training**: Multiple models are trained on different views of the data. Each model labels the unlabeled data, and these labels are used to train the other models.
    - **Generative Models**: Models like variational autoencoders (VAEs) or generative adversarial networks (GANs) generate synthetic data to augment the training process.

Key Differences:

- **Label Availability**:
    - **Unsupervised Learning**: No labeled data is available. The model learns from the structure and distribution of the data.
    - **Semi-Supervised Learning**: Both labeled and unlabeled data are available. The### Detailed Answers to Homework Questions