

CIFAR-100 Analysis Documentation

Overview

This documentation describes the implementation of an automated image selection and analysis system using CNN features and SBERT semantic embeddings. The system selects a diverse subset of images from the CIFAR-100 dataset using hierarchical clustering on CNN features, then compares these selections with their semantic relationships.

Implementation Details

1. Feature Extraction and Loading

```
def load_features(feature_dir='cifar_features'):  
    """  
    Loads three types of data:  
    - CNN features extracted from AlexNet  
    - SBERT semantic embeddings  
    - Metadata containing class names and indices  
    """
```

The system uses:

- CNN features from AlexNet's last maxpool layer
- Semantic embeddings from SBERT
- JSON metadata for tracking image information

2. Similarity Computation

```
def compute_similarity_matrices(cnn_features, semantic_features):  
    """  
    Computes two similarity matrices:  
    - CNN-based similarities using cosine similarity  
    - SBERT-based similarities using cosine similarity  
    """
```

Key aspects:

- Uses cosine similarity for both feature types
- Ranges from -1 (opposite) to 1 (identical)
- Handles different feature dimensionalities

3. Image Selection Process

```
def select_diverse_images(similarity_matrix, n_select=20, method='ward'):
    """
    Selects diverse images using hierarchical clustering
    """
```

Selection process:

1. Convert similarities to distances (1 - similarity)
2. Perform hierarchical clustering using Ward's method
3. Cut tree to get n_select clusters
4. Select representative image from each cluster (closest to center)

4. Visualization Components

a. Full Dataset Visualization

```
def visualize_clustering(similarity_matrix, metadata, selected_indices):
    """
    Creates two visualizations:
    1. Dendrogram showing hierarchical relationships
    2. Heatmap showing similarity matrix with selected images highlighted
    """
```

b. Comparison Visualization

```
def visualize_comparison(cnn_similarity, sbert_similarity, metadata,
selected_indices):
    """
    Creates four visualizations:
    1. CNN-based dendrogram
    2. SBERT-based dendrogram
    3. CNN similarity heatmap
    4. SBERT similarity heatmap
    """
```

c. Pair Analysis

```
def analyze_pairs(similarity_matrix, metadata, selected_indices, top_n=10):
    """
    Analyzes and reports:
    - Top N most similar pairs
    - Top N least similar pairs
    """
```

5. Main Workflow

1. Data Loading

- Load CNN features
- Load SBERT embeddings
- Load metadata

2. Selection Process

- Compute CNN similarity matrix
- Select diverse images using CNN features only
- Number of images configurable (default: 20)

3. Analysis

- Visualize full dataset clustering
- Show selected images in both CNN and SBERT space
- Analyze most/least similar pairs

4. Output

- Prints selected image information
- Saves results to JSON file
- Generates visualizations

6. JSON Output Format

```
{
  "selected_indices": [...],
  "selected_images": [
    {
      "class_name": "...",
      "index": ...,
      "feature_file": "..."
    },
    ...
  ],
  "cnn_most_similar_pairs": [...],
  "cnn_least_similar_pairs": [...]
}
```

Usage

1. Basic Usage

```
python cifar_analysis.py
```

2. Configuration Options

- Adjust `n_select` in `main()` for different subset sizes
- Modify visualization parameters for different views
- Change clustering method in `select_diverse_images()`

Key Features

1. Automated Selection

- Uses hierarchical clustering for diverse image selection
- Selects representative images from each cluster
- Configurable number of selections

2. Dual Analysis

- CNN-based visual feature analysis
- SBERT-based semantic analysis
- Direct comparison between both spaces

3. Comprehensive Visualization

- Full dataset structure
- Selected subset relationships
- Side-by-side CNN vs SBERT comparison

4. Detailed Reporting

- Most similar image pairs
- Least similar image pairs
- Full selection metadata

Implementation Notes

1. Clustering Method

- Uses Ward's method for hierarchical clustering
- Minimizes variance within clusters
- Produces more balanced clusters

2. Similarity Metrics

- Cosine similarity for both CNN and SBERT
- Ranges from -1 to 1
- Diagonal masked in visualizations

3. Visualization Design

- Clear separation of CNN and SBERT results
- Highlighted selected images in full dataset view
- Detailed labels for selected subset

Future Improvements

1. Scalability

- Optimize for larger datasets
- Implement batch processing
- Add memory-efficient options

2. Visualization

- Add interactive visualization options
- Implement zoom capabilities for large datasets
- Add export options for high-resolution figures

3. Analysis

- Add statistical measures of selection quality
- Implement alternative selection methods
- Add cross-validation options

Dependencies

- PyTorch
- torchvision
- sentence-transformers
- numpy
- json

CIFAR-100 Dataset

Dataset Overview

CIFAR-100 is a widely-used computer vision dataset consisting of 60,000 color images (32x32 pixels) in 100 different classes. The dataset is split into:

- 50,000 training images
- 10,000 test images
- 100 classes with 600 images per class
- 20 superclasses, each containing 5 related classes

Why CIFAR-100 for Clustering Validation

CIFAR-100 is ideal for validating our CNN clustering approach for several reasons:

1. Diverse Content

- 100 distinct classes covering a wide range of objects and concepts
- Natural variations within each class
- Multiple related classes within superclasses

2. Controlled Environment

- Consistent image size and format
- Well-labeled and categorized
- Known relationships between classes

3. Scale Testing

- Large enough to test scalability
- Small enough to process efficiently
- Good balance of diversity and manageability

Cluster-Based Selection Algorithm

Selection Process Details

1. Hierarchical Clustering

```
def select_diverse_images(similarity_matrix, n_select=20, method='ward'):
    # Convert similarities to distances
    distances = 1 - similarity_matrix
    np.fill_diagonal(distances, 0)

    # Create condensed distance matrix
    condensed_distances = squareform(distances)

    # Perform hierarchical clustering
    linkage_matrix = linkage(condensed_distances, method='ward')

    # Cut tree to get n_select clusters
    clusters = fcluster(linkage_matrix, n_select, criterion='maxclust')
```

2. Representative Selection

```
# For each cluster
for i in range(1, n_select + 1):
    cluster_indices = np.where(clusters == i)[0]

    # Find image closest to cluster center
    cluster_similarities = similarity_matrix[cluster_indices][:,
cluster_indices]
    mean_similarities = np.mean(cluster_similarities, axis=1)
    center_idx = cluster_indices[np.argmax(mean_similarities)]

    selected_indices.append(center_idx)
```

Algorithm Explanation

1. Distance Calculation

- Converts similarity scores to distances using `1 - similarity`

- Higher similarity = smaller distance
- Range: 0 (identical) to 2 (opposite)

2. Ward's Method

- Uses Ward's minimum variance criterion
- Minimizes within-cluster variance
- Creates compact, well-balanced clusters

3. Cluster Cutting

- Cuts dendrogram to create exactly n_{select} clusters
- Uses maxclust criterion to ensure desired number of clusters
- Maintains maximum diversity between clusters

4. Center Selection

- For each cluster:
 1. Identifies all images in the cluster
 2. Computes mean similarity to all other images in cluster
 3. Selects image with highest mean similarity (closest to center)
- This ensures selected image is most representative of its cluster

Why This Approach Works

1. Diversity Guarantee

- Each cluster represents a distinct region in feature space
- Selecting one image per cluster ensures diversity
- Ward's method prevents creation of very small or large clusters

2. Representative Selection

- Center-based selection ensures typical examples
- Avoids outliers or edge cases
- Maintains cluster coherence

3. Scalability

- Works efficiently with large datasets
- Automatically determines optimal clustering
- No manual parameter tuning needed

Example Selection Process

```
Initial Dataset: 600 images per class × 100 classes = 60,000 images
↓
Compute CNN Features: 60,000 feature vectors
↓
Build Similarity Matrix: 60,000 × 60,000 matrix
↓
```

```
Hierarchical Clustering: Cut into n_select clusters
↓
Select Centers: One representative per cluster
↓
Final Output: n_select diverse, representative images
```

This approach ensures that the selected images:

- Are maximally different from each other (diversity)
- Represent their local image neighborhoods well (representativeness)
- Cover the full range of visual concepts in the dataset (coverage)