Dictionary Learning on Image Patches - Complete Guide

📋 Project Overview

This project demonstrates **sparse dictionary learning** using scikit-learn on image data. Dictionary learning is an unsupervised machine learning technique that discovers a compact set of basis functions (called "atoms") that can efficiently represent image patches through sparse linear combinations.

What is Dictionary Learning?

Dictionary learning aims to find a dictionary **D** and sparse codes **α** such that:

$$X \approx \alpha \cdot D$$

Where:

- **X** is the input data (image patches)
- **D** is the learned dictionary (basis functions/atoms)
- **α** is the sparse representation (most coefficients are zero)

This is particularly useful for:

- **Image compression** (fewer coefficients needed)
- **Feature extraction** for machine learning
- **Image denoising** and restoration
- **Understanding natural image statistics**

🗂️ Files Generated

1. Visualizations (PNG files)

01_dictionary_atoms.png

- **What it shows**: All 100 learned dictionary atoms arranged in a 10×10 grid
- **Interpretation**: Each small image represents a "basis function" or pattern that the algorithm discovered in the dataset
- **What to look for**: Edges, textures, gradients, and repetitive patterns
- **Size**: 111 KB

02_reconstruction_comparison.png

- **What it shows**: Side-by-side comparison of 10 original patches (top row) vs. reconstructed patches (bottom row)
- **Interpretation**: Demonstrates how well the sparse representation can reconstruct the original data
- **Quality metric**: Closer visual similarity = better dictionary learning
- **Size**: 39 KB

03_sparse_codes_examples.png

- **What it shows**: Bar plots of sparse coefficients for 5 example patches
- **Interpretation**: Each bar chart shows which dictionary atoms (x-axis) are used to represent a patch and their weights (y-axis)
- **Sparsity**: Most bars are zero, demonstrating sparse representation
- **Size**: 46 KB

04_sparsity_histogram.png

- **What it shows**: Distribution of sparsity across all 1000 sample patches
- **Interpretation**: Shows what percentage of dictionary atoms are actually used per patch
- **Key finding**: Mean sparsity of 2.58% means each patch uses only ~2-3 atoms out of 100
- **Size**: 50 KB

---

2. Numerical Data (NPY files)

These are NumPy arrays that can be loaded in Python with `np.load('filename.npy')`

dictionary.npy

- **Shape**: (100, 256)
- **Content**: The learned dictionary matrix
  - 100 rows = 100 dictionary atoms
  - 256 columns = flattened 16×16 patches
- **Usage**: `dictionary = np.load('dictionary.npy')`
- **Size**: 201 KB

sparse_codes_sample.npy

- **Shape**: (1000, 100)
- **Content**: Sparse representation coefficients
  - 1000 rows = 1000 sample patches

    - 100 columns = coefficients for each dictionary atom
- **Usage**: `sparse_codes = np.load('sparse_codes_sample.npy')`
- **Note**: Most values are zero (sparse!)
- **Size**: 782 KB

reconstructed_patches.npy

- **Shape**: (1000, 16, 16)
- **Content**: Patches reconstructed from sparse codes
- **Formula**: reconstructed = sparse_codes @ dictionary
- **Usage**:

```python
reconstructed = np.load('reconstructed_patches.npy')
plt.imshow(reconstructed[0], cmap='gray')
```

- **Size**: 2.0 MB

original_patches_sample.npy

- **Shape**: (1000, 16, 16)
- **Content**: Original patches before sparse coding
- **Usage**: Compare with reconstructed_patches.npy
- **Size**: 2.0 MB

---

3. Machine Learning Model (PKL file)

dict_learning_model.pkl

- **Content**: Trained scikit-learn MiniBatchDictionaryLearning object
- **Usage**:

```python
import pickle
with open('dict_learning_model.pkl', 'rb') as f:
    model = pickle.load(f)

# Transform new patches
new_sparse_codes = model.transform(new_patches)
```

- **Size**: 485 KB

---

4. Summary Reports (TXT files)

`RESULTS_SUMMARY.txt`

- **Content**: Complete summary of the experiment including parameters, results, and interpretation
- **Includes**: Dataset info, algorithm parameters, sparsity statistics, file descriptions
- **Size**: 2.3 KB

📊 Key Results

Dataset Statistics

- **Images processed**: 300 synthetic images (256×256 pixels)
- **Patches extracted**: 15,000 patches (16×16 pixels each)
- **Image types**: Various patterns including gradients, Gabor filters, geometric shapes, and textures

Dictionary Learning Parameters

- **Dictionary size**: 100 atoms
- **Sparsity parameter (α)**: 1.0
- **Algorithm**: LARS (Least Angle Regression)
- **Iterations**: 100
- **Batch size**: 200

Performance Metrics

- **Mean sparsity**: 2.58% (average of 2-3 non-zero coefficients per patch)
- **Median sparsity**: 2.00%
- **Reconstruction error (MSE)**: 0.007608
- **Interpretation**: Very sparse representation with low reconstruction error = excellent performance!

🔬 How to Use These Results

1. Visualize Dictionary Atoms

```
python
```

```python
import numpy as np
import matplotlib.pyplot as plt

# Load dictionary
dictionary = np.load('dictionary.npy')

# Visualize a single atom
atom_idx = 0
atom = dictionary[atom_idx].reshape(16, 16)
plt.imshow(atom, cmap='gray')
plt.title(f'Dictionary Atom {atom_idx}')
plt.colorbar()
plt.show()
```

2. Reconstruct Images from Sparse Codes

```python
# Load data
dictionary = np.load('dictionary.npy')
sparse_codes = np.load('sparse_codes_sample.npy')
originals = np.load('original_patches_sample.npy')

# Reconstruct a specific patch
patch_idx = 5
reconstructed = sparse_codes[patch_idx] @ dictionary
reconstructed = reconstructed.reshape(16, 16)

# Compare
fig, (ax1, ax2) = plt.subplots(1, 2)
ax1.imshow(originals[patch_idx], cmap='gray')
ax1.set_title('Original')
ax2.imshow(reconstructed, cmap='gray')
ax2.set_title('Reconstructed')
plt.show()
```

3. Transform New Patches

```python
```

```python
import pickle

# Load trained model
with open('dict_learning_model.pkl', 'rb') as f:
    model = pickle.load(f)

# Transform new patches (must be 16x16, flattened to 256-dim vectors)
new_patches = your_patches.reshape(-1, 256)
new_sparse_codes = model.transform(new_patches)

# Reconstruct
reconstructed = model.transform(new_patches) @ model.components_
```

4. Analyze Sparsity

```python
sparse_codes = np.load('sparse_codes_sample.npy')

# Count non-zero coefficients per patch
non_zero_counts = np.sum(sparse_codes != 0, axis=1)

print(f"Min non-zero coefficients: {non_zero_counts.min()}")
print(f"Max non-zero coefficients: {non_zero_counts.max()}")
print(f"Mean non-zero coefficients: {non_zero_counts.mean():.2f}")

# Which atoms are most frequently used?
atom_usage = np.sum(sparse_codes != 0, axis=0)
top_atoms = np.argsort(atom_usage)[::-1][:10]
print(f"Most used atoms: {top_atoms}")
```

🧪 Experiment Ideas

1. Change Sparsity Level

Try different alpha values in the original script:

- **α = 0.1**: Less sparse (more atoms used per patch)
- **α = 1.0**: Current setting
- **α = 5.0**: More sparse (fewer atoms used per patch)

2. Different Dictionary Sizes

Modify `n_components`:

- **50 atoms**: Smaller dictionary, faster training
- **100 atoms**: Current setting
- **200 atoms**: Larger dictionary, more expressive

3. Different Patch Sizes

Change `patch_size`:

- **8×8**: Smaller patches, faster processing
- **16×16**: Current setting
- **32×32**: Larger patches, more context

4. Real Image Datasets

Replace synthetic images with real datasets:

- **CIFAR-10**: Natural images (if network available)
- **Your own images**: Custom dataset
- **Medical images**: Specialized domain

📚 Understanding the Algorithm

Step-by-Step Process

1. **Patch Extraction** (Step 2)
   - Randomly extract 16×16 patches from 256×256 images
   - Each patch becomes a 256-dimensional vector
   - Center the data (subtract mean)
2. **Dictionary Learning** (Step 3)
   - Initialize 100 random dictionary atoms
   - Iteratively:
     - **Sparse coding**: Find sparse coefficients for current dictionary
     - **Dictionary update**: Update dictionary to minimize reconstruction error
   - Uses LARS algorithm for efficiency
3. **Sparse Representation** (Step 4)
   - Transform patches into sparse coefficients
   - Most coefficients are zero (sparsity)
   - Only 2-3 atoms needed on average

4. **Reconstruction** (Step 5)
    - Multiply sparse codes by dictionary
    - Compare with originals
    - Measure reconstruction error

Mathematical Formulation

Optimization objective:

$$\text{minimize } ||X - \alpha D||^2 + \lambda||\alpha||_1$$

Where:

- $||\cdot||^2$: Squared reconstruction error
- $||\cdot||_1$: L1 norm (promotes sparsity)
- λ (alpha): Sparsity-regularization tradeoff

🎯 Applications

Image Processing

- **Compression**: Store images with sparse coefficients instead of pixels
- **Denoising**: Represent clean signal, filter out noise
- **Inpainting**: Fill in missing regions using dictionary

Machine Learning

- **Feature extraction**: Use sparse codes as features for classification
- **Transfer learning**: Apply learned dictionary to new tasks
- **Anomaly detection**: Unusual patterns have poor reconstruction

Computer Vision

- **Edge detection**: Dictionary atoms often resemble edge filters
- **Texture analysis**: Atoms capture texture patterns
- **Object recognition**: Sparse codes describe local features

---

📖 References

Scikit-learn Documentation

- MiniBatchDictionaryLearning
- Dictionary Learning Tutorial

Research Papers

- Olshausen & Field (1996): "Emergence of simple-cell receptive field properties by learning a sparse code for natural images"
- Mairal et al. (2010): "Online Learning for Matrix Factorization and Sparse Coding"

Related Algorithms

- **PCA**: Linear dimensionality reduction (not sparse)
- **ICA**: Independent component analysis
- **NMF**: Non-negative matrix factorization
- **Sparse PCA**: PCA with sparsity constraints

🛠️ Requirements

To run the code or use these results:

```bash
pip install numpy scikit-learn matplotlib scipy
```

Or with the break-system-packages flag if needed:

```bash
pip install numpy scikit-learn matplotlib scipy --break-system-packages
```

💡 Tips for Interpretation

What Makes Good Dictionary Atoms?

- **Diverse**: Cover many different patterns
- **Localized**: Capture local features
- **Interpretable**: Resemble edges, textures, or patterns

- **Efficient**: Few atoms reconstruct many patches

Signs of Good Performance

✅ Low reconstruction error
✅ High sparsity (most coefficients are zero)
✅ Visually interpretable dictionary atoms
✅ Good reconstruction quality
✅ Diverse atom patterns

Signs of Poor Performance

❌ High reconstruction error
❌ Low sparsity (many coefficients needed)
❌ Noisy or unintelligible atoms
❌ Many unused atoms
❌ Blurry reconstructions

📎 Files Summary Table

File	Type	Size	Description
01_dictionary_atoms.png	Image	111 KB	Visualization of 100 learned atoms
02_reconstruction_comparison.png	Image	39 KB	Original vs reconstructed patches
03_sparse_codes_examples.png	Image	46 KB	Bar plots of sparse coefficients
04_sparsity_histogram.png	Image	50 KB	Sparsity distribution
dictionary.npy	NumPy	201 KB	Dictionary matrix (100×256)
sparse_codes_sample.npy	NumPy	782 KB	Sparse coefficients (1000×100)
reconstructed_patches.npy	NumPy	2.0 MB	Reconstructed patches
original_patches_sample.npy	NumPy	2.0 MB	Original patches
dict_learning_model.pkl	Pickle	485 KB	Trained sklearn model
RESULTS_SUMMARY.txt	Text	2.3 KB	Results summary

Total size: ~5.6 MB

📪 Questions?

For questions about dictionary learning or this implementation:

- Check the scikit-learn documentation
- Review the commented source code
- Experiment with different parameters
- Compare with published research papers

Happy learning! 🚀