Tiny ImageNet-15 Classification with Bag of Visual Words and Neural Networks

Project Summary:

- This project implements an image classification pipeline on a 15-class subset of the Tiny ImageNet dataset. It combines traditional feature-based methods and deep learning models.
- First, we extract local features from each image using SIFT (Scale-Invariant Feature Transform) and ORB (Oriented FAST and Rotated BRIEF) en.wikipedia.orgdocs.opencv.org. These features are then quantized into a *Bag of Visual Words* (BoVW) representation via K-Means clustering pinecone.iopinecone.io.
- Two classifiers are trained on this BoVW representation: (1) a feed-forward neural network (MLP) implemented with scikit-learn's MLPClassifier scikit-learn.org, and (2) a convolutional neural network (CNN) implemented in PyTorch datacamp.com. The CNN is trained directly on the raw images (with convolution, pooling, and fully connected layers).
- The performance of both models is evaluated using overall accuracy as well as detailed metrics. We compute confusion matrices and classification reports (precision, recall, F1-score) using scikit-learn to analyse per-class performance scikit-learn.orgscikit-learn.org.

Environment and Tools:

- Development is done in a Jupyter Notebook within an Anaconda Python 3 environment.
- Key libraries include OpenCV for feature extraction, NumPy for array operations, scikit-learn for the MLP classifier and evaluation metrics, PyTorch (and torchvision) for the CNN, Matplotlib for plotting, and tqdm for progress bars.
- The code is written for local execution; GPU acceleration can be used for CNN training if available.

Dataset Details:

- We use a subset of the Tiny ImageNet dataset. Tiny ImageNet is a reduced version of the ImageNet dataset, containing 100,000 training images across 200 classes, with each image scaled to 64×64 pixels datasets.activeloop.aidatasets.activeloop.ai. In this project, we select 15 classes from Tiny ImageNet to form our working dataset.
- Each class has 500 training images (64×64 color) and 50 validation images. The images are organized in class-specific folders. We resize or normalize images as needed during preprocessing.

Feature Extraction (SIFT and ORB):

- We detect and describe key points in each image using two methods: **SIFT** and **ORB**. SIFT is a scale- and rotation-invariant feature descriptor that produces a 128-dimensional descriptor for each keypoint en.wikipedia.org. ORB is an efficient, open- source alternative combining FAST key point detection with an oriented BRIEF descriptor docs.opencv.org.

- For each image, both SIFT and ORB are applied (using OpenCV). The resulting descriptor vectors from all training images form the basis for building the BoVW codebook.

Feature Encoding (Bag of Visual Words):

- We aggregate all SIFT/ORB descriptors from the training set and apply K-Means clustering to form a visual vocabulary (codebook) of size *k* pinecone.io. Each cluster centre represents a "visual word."
- Each image is then encoded as a histogram of visual word occurrences: for every descriptor in the image, we find the nearest cluster centroid and increment its histogram bin. This yields a fixed-length BoVW feature vector per image pinecone.io.
- The BoVW histograms are used as input features for the MLP classifier.

Model Architectures and Training:

- **MLP (scikit-learn):** We use sklearn.neural_network.MLPClassifier to train a feed-forward neural network on the BoVW features scikit-learn.org. In our implementation, the MLP has one hidden layer (e.g. 100 neurons) with ReLU activations, and is trained using the Adam optimizer (default settings). The output layer has 15 units (one per class) with softmax. Training is done by minimizing the log-loss (cross-entropy) on the training set.
- **CNN (PyTorch):** We implement a convolutional neural network using PyTorch datacamp.com. The architecture consists of multiple convolutional layers (with ReLU and max-pooling) followed by one or more fully connected layers. The network takes 64×64 images as input and outputs class probabilities for 15 classes. We use nn.CrossEntropyLoss and the Adam optimizer. The CNN is trained for a fixed number of epochs (e.g. 20–30) with a suitable batch size.

Evaluation Results:

- After training, each model is evaluated on a held-out test/validation set of images. We report the overall accuracy score for each model. For detailed analysis, we generate a **confusion matrix** and a **classification report** (precision, recall, F1-score per class) using scikit-learn's metrics scikit-learn.orgscikit-learn.org.
- These reports help identify which classes are confused and how balanced the predictions are. In general, the CNN typically achieves higher accuracy by learning rich spatial features, while the BoVW+MLP pipeline provides a solid baseline using hand-crafted features.

Instructions to Run:

1. **Environment setup:** Create a Python 3 (e.g. Anaconda) environment and install the required libraries (see below or requirements.txt).
2. **Dataset preparation:** Download the Tiny ImageNet dataset and extract the images for the 15 chosen classes into the notebook's working directory, preserving the train/val folder structure. Ensure images are accessible by the notebook.
3. **Run the notebook:** Launch Jupyter Notebook and open the project notebook. Run all cells in order. The notebook performs resizing (if needed), feature extraction, BoVW encoding, model training, and evaluation.

4. **View outputs:** Classification accuracy, confusion matrices, and classification reports will be displayed or plotted in the notebook. You can re-run with different parameters (e.g. number of BoVW clusters, MLP hidden units, CNN layers, etc.) as experiments.

Dependencies:
Required Python packages (can be installed via pip or conda) include:

- opencv-python (OpenCV) for feature extraction
- numpy for numerical computations
- scikit-learn for MLPClassifier, KMeans, and metrics
- torch and torchvision for the CNN model
- matplotlib for plotting results
- tqdm for progress bars
- jupyter (or notebook) to run the Jupyter Notebook

See the accompanying requirements.txt file for exact package names.

Precomputed Feature Vectors

For ease of use, the precomputed feature vectors are available in the GitHub Releases section:

[fisher_test]
[fisher_train]
[fisher_vectors]

Author: Mohammad Arsalan