B31MV Lab2 (Week 5)

11th February, 2025

Google Colab Notebook:
https://colab.research.google.com/drive/1QEdiVbo16g6J5JpDvLkWhGHNjLGSVmIr?usp=sharing

1 Bag of Visual Words (BoVW)

What is Bag of Visual Words (BoVW)?

BoVW is a feature extraction technique used in image classification. It treats local features in an image (e.g., edges, corners) like words in a document and builds a "visual vocabulary." Images are then represented as histograms of these visual words, which can be fed into a machine learning model for classification.

This section explains the implementation of the **Bag of Visual Words (BoVW)** approach for image classification, along with the reasoning behind each step and the significance of the parameter values used.

Step 1: Dataset Preparation

- **Task:** Extract the dataset `101_ObjectCategories.tar.gz` to load images.
- **Why:** A structured dataset is essential for categorizing images into predefined classes.
- **Key Arguments:**
  - `archive_path`: Path to the dataset archive file.
  - `dataset_extract_path`: Path where the dataset is extracted.

  The dataset is divided into folders representing categories (e.g., airplanes, faces).

Step 2: Preprocessing Images

- **Task:** Load images, convert them to grayscale, and resize them to `150x150` pixels.
- **Why:** Grayscale reduces computational complexity, and resizing ensures uniform input size for feature extraction.
- **Key Arguments:**
  - `image_size = (150, 150)`: Fixed size to standardize image dimensions.

Step 3: Visualizing Sample Images

- **Task:** Display one image per category to understand the dataset visually.
- **Why:** Visual inspection verifies the dataset structure and category distribution.

Step 4: Splitting the Dataset

- **Task:** Divide the dataset into **training (80%)** and **testing (20%)** sets.
- **Why:** Separate data for training and evaluation ensures unbiased testing.
- **Key Arguments:**
  - `test_size = 0.2`: Fraction of the dataset used for testing.

Step 5: Feature Extraction using ORB

- **Task:** Extract features from images using **ORB (Oriented FAST and Rotated BRIEF)**.
- **Why:** ORB detects keypoints and computes descriptors for local image regions, providing robust features for clustering.
- **Key Arguments:**
  - `nfeatures = 500`: Number of keypoints to detect per image. A higher value captures more detail but increases computational cost.

Step 6: Clustering Descriptors (Visual Vocabulary)

- **Task:** Cluster descriptors into **100 visual words** using **K-Means**.
- **Why:** Clustering groups similar features, creating a visual vocabulary for representing images.
- **Key Arguments:**
  - `n_clusters = 100`: Number of visual words. Higher values provide finer granularity but require more data and computation.

Step 7: Visualizing Clustering Using PCA

- **Task:** Use **Principal Component Analysis (PCA)** to reduce descriptors to 2D and plot clusters.
- **Why:** Visualizing clustering verifies that descriptors are meaningfully grouped.
- **Key Arguments:**
  - `n_components = 2`: Reduces descriptor space to two dimensions for visualization.

Step 8: Converting Images to Histograms

- **Task:** Represent each image as a histogram of visual words.
- **Why:** Histograms summarize the frequency of visual words, enabling comparison between images.

Step 9: Training an SVM Classifier

- **Task:** Train a Support Vector Machine (SVM) classifier using the histograms.
- **Why:** SVMs are effective for high-dimensional data and work well for image classification.
- **Key Arguments:**
  - `kernel = 'linear'`: Linear kernel for a simple decision boundary.

Step 10: Evaluating the Model

- **Task:** Test the classifier on unseen images and calculate accuracy.
- **Why:** Evaluates the model's ability to generalize to new data.

Step 11: Testing on an Individual Image

- **Task:** Predict the category of a single image using the trained classifier.
- **Why:** Demonstrates the model's real-world application.
- **Key Arguments:**
  - Input image path: Path to the image being tested.

2 Supervised Learning for Image Classification

What is Supervised Learning and Image Classification?

Supervised learning is a paradigm of machine learning where a model learns from labeled data. **Image classification** is the task of assigning a label or category to an image based on its visual content. Within the context of supervised learning, the image classification task involves a labeled dataset (each image is associated with a known label), and a model that learns to map input images to correct output categories.

This section explains the implementation of a **supervised image classification task** on the MNIST dataset (contains images of handwritten numerals) using classic machine learning algorithms (KNN, Random Forest, Logistic Regression) and a deep learning (CNN) method. In this section, evaluation metrics such as accuracy (how many total predictions are correct), precision (out of all the predicted positive cases, how many are actually correct), recall (out of all actual positive cases, how many did are correctly predicted) and f1-score (balances both precision and recall using their harmonic mean) are used.

Step 2: Dataset Preprocessing

- **Task:** Perform preprocessing tasks on the imported dataset.
- **Why:** The dataset needs to be split and transformed (flattened, one-hot encoded) to be used for the image classification task.
- **Key Arguments:**
  - `X_train.reshape(X_train.shape[0], -1)`: Flatten input data. Classical methods like Logistic regression , random forest etc. require 1D feature vectors as input rather than 2D images.
  - `to_categorical()`: Performs one-hot encoding on output data. One-hot encoding converts categorical labels (0–9 digits) into binary vectors, which is compactible with the softmax activation layer (output layer) in some deep learning methods.
  - `X_train / 255.0, X_test / 255.0`: Normalizes data. Scales pixel values to a standard range, typically between 0 and 1 . The value 255.0 is used due to grayscale range of 0-255.0.

Step 3: Dataset Visualization

- **Task:** Visualize one image per category from the dataset.
- **Why:** Visual inspection verifies the dataset structure and category distribution.

Step 4: Logistic Regression

- **Task:** Train a Logistic Regression Model on the dataset and evaluate its predictions.
- **Why:** Logistic Regression is a classification algorithm used to predict categorical outcomes.
- **Key Arguments:**
  - `max_iter = 1000`: Maximum iterations the optimization algorithm will run before stopping.

Step 5: Random Forest

- **Task:** Train a Random Forest Model on the dataset and evaluate its predictions.
- **Why:** Random Forest is a powerful classification algorithm that builds multiple decision trees and combines their outputs, reducing the risk of overfitting.
- **Key Arguments:**
  - `n_estimators=100`: Specifies the number of decision trees in the ensemble.

Step 6: KNN

- **Task:** Train a KNN Model on the dataset and evaluate its predictions.
- **Why:** KNN is a learning algorithm that classifies new data based on similarity to stored examples.
- **Key Arguments:**
  - `n_neighbors=5`: Specifies the number of nearest neighbors when making a prediction.

Step 7: Deep Learning (CNN)

- **Task:** Train a CNN model on the dataset, evaluate its predictions, and visualize both a confusion matrix (compares the actual labels with the predicted labels, showing the number of correct and incorrect predictions for each class) and random sample predictions from the trained model.
- **Why:** CNN are powerful models that can be used for image classification to process and learn from spatial and hierarchical patterns in images.
- **Key Arguments:**
  - `Conv2D(32, (3,3), activation='relu')`: Convolutional layer with 32 filters, 3×3 kernel size, and ReLU activation.
  - `MaxPooling2D((2,2))`: Max-pooling layer with a 2×2 filter to downsample the feature maps.
  - `Flatten()`: Flattens the 2D feature maps into a 1D vector for the dense layers.
  - `Dense(128, activation='relu')`: Fully connected layer with 128 neurons and ReLU activation.
  - `Dropout(0.5)`: Dropout regularization (50%) to prevent overfitting.
  - `Dense(10, activation='softmax')`: Output layer with 10 neurons (for 10 classes) and softmax activation for probability distribution.
  - `optimizer='adam'`: Adam optimizer, an adaptive learning rate optimization algorithm.

- `loss='categorical_crossentropy'`: Loss function used for multi-class classification.
- `metrics=['accuracy'])`: Evaluates model performance based on classification accuracy.
- `epochs=5`: The model trains for 5 iterations over the entire dataset.
- `batch_size=64`: The model updates weights after processing 64 images per batch.
- `validation_data=(X_test_cnn, y_test_onehot)`: Validation set to monitor performance during training.
- `np.argmax(y_pred_prob, axis=1)`: Converts probabilities into predicted class labels.
- `y_pred_prob = cnn_model.predict(X_test)`: CNN outputs probabilities for each class.

Step 8: Model Comparison

- **Task:** Compare the evaluation metrics (accuracy, precision, recall and F1-score) for the trained models.
- **Why:** To identify the best model.