Extract Bag of Words Features and Match Images

Introducere

Scopul proiectului este de a dezvolta un sistem de regăsire a imaginilor bazat pe metoda Bag of Visual Words, folosind dataset-ul 15-Scene. Metoda transformă imaginile, inițial reprezentate ca simple matrici de pixeli în vectori de caracteristici (histograme), care pot fi apoi comparate pentru a identifica imagini similare.

Specificații Detaliate

Formatul datelor de intrare și de ieșire

Proiectul se bazează pe dataset-ul 15-Scene, care include imagini color organizate în 15 clase de scene naturale și artificiale (dormitor, bucătărie, salon, magazin de cărți, etc.). Dataset-ul conține între 200-400 de imagini per clasă, cu dimensiuni variabile care vor fi redimensionate la 256×256 pixeli pentru procesare uniformă. Imaginile vor fi încărcate în aplicație fie ca fișiere individuale organizate pe directoare, fie ca liste de imagini însoțite de etichete, iar reprezentarea internă va fi realizată sub formă de matrici de pixeli.

Prezentarea temei și obiectivele proiectului

Se dorește dezvoltarea unui sistem capabil să extragă caracteristici vizuale din imagini și să le transforme într-o reprezentare vectorială fixă (histogramă) utilizând metoda Bag of Visual Words. Acest proces implică construirea unui vocabular vizual, prin aplicarea unui algoritm de clustering (k-means) asupra descriptorilor locali extrași din imagini, și apoi reprezentarea fiecărei imagini ca o histogramă ce indică frecvența cuvintelor vizuale. Obiectivele proiectului sunt:

- Extragerea caracteristicilor locale folosind algoritmul SIFT
- Construirea unui vocabular vizual prin clustering (k-means)
- Generarea histogramelor Bag of Words pentru fiecare imagine
- Potrivirea imaginilor pe baza comparării histogramelor
- Salvarea modelului (vocabularul și histogramele) pentru reutilizare ulterioară

Detalierea cerințelor funcționale ale aplicației

Funcționalitatea aplicației se împarte în mai multe etape. În primul rând, se va încărca dataset-ul 15-Scene și se vor preprocesa imaginile, astfel încât acestea să fie într-un format potrivit pentru analiză. Ulterior, se va aplica algoritmul SIFT pentru a detecta punctele de interes și a calcula descriptorii locali ai fiecărei imagini. Acești descriptori vor fi apoi agregați și folosiți pentru a construi un vocabular vizual prin intermediul algoritmului de clustering k-means. Odată definit vocabularul, fiecare imagine va fi reprezentată printr-o histogramă Bag of Words, calculată prin cuantizarea descriptorilor în cuvinte vizuale. În final, pentru o imagine interogare, aplicația va calcula histograma și o va compara cu cele existente, returnând imaginile din baza de date care au cel mai mare grad de similaritate. Această

abordare permite transformarea unei probleme complexe (compararea imaginilor brute) într-o operație de comparare între vectori de dimensiune fixă.

Analiză și Fundamentare Teoretică

Din punct de vedere teoretic, metoda Bag of Visual Words tratează o imagine la fel cum un document text este reprezentat printr-un sac de cuvinte. În cazul imaginilor, cuvintele sunt reprezentate de descriptorii locali extrași din puncte de interes, iar distribuția acestor descriptorii într-un spațiu de caracteristici este redusă la o histogramă. Această histogramă, care are o dimensiune fixă egală cu mărimea vocabularului vizual, servește drept reprezentare a conținutului imaginii și permite compararea directă între imagini prin măsuri standard de similaritate sau distanță.

Algoritmi utilizați

Pentru extragerea caracteristicilor, se va folosi algoritmul SIFT, care detectează punctele de interes și calculează descriptorii locali. Acești descriptori, care au de obicei 128 de componente, sunt invariabili la transformări de scară, rotație și schimbări de iluminare, ceea ce îi face potriviți pentru probleme de recunoaștere a imaginilor. Pentru construirea vocabularului vizual, se utilizează algoritmul de clustering k-means. Acesta grupează descriptorii în k clustere, unde fiecare cluster este reprezentat prin centrul său, devenind astfel un „cuvânt vizual". Fiecare imagine este apoi reprezentată printr-o histogramă ce indică numărul de descriptorii care au fost asociați fiecărui cuvânt vizual.

Explicații și argumentări logice ale soluției alese

Soluția propusă are valoare teoretică și practică datorită faptului că transformă imagini complexe în vectori de caracteristici de dimensiune fixă, facilitând astfel compararea acestora. Prin folosirea descriptorilor SIFT, se asigură că trăsăturile esențiale ale imaginilor sunt captate în mod robust, iar prin aplicarea algoritmului k-means se obține un vocabular care sintetizează diversitatea vizuală din dataset. Normalizarea histogramelor asigură o comparabilitate uniformă între imagini, iar utilizarea măsurilor de similaritate, cum ar fi similaritatea cosinus sau distanța Euclidiană, permite identificarea rapidă a imaginilor similare. Astfel, întregul sistem poate fi înțeles ca o succesiune de pași care transformă datele brute (imagini) într-un format abstract și compact, ce reflectă esența vizuală a fiecărei imagini.

Structura logică și funcțională a aplicației

Am implementat aplicația în C++ folosind OpenCV, împărțind codul în 7 pași principali care se execută secvențial. Fiecare pas are o funcție dedicată care se ocupă de o etapă specifică din procesul de clustering.

Configurarea inițială și constantele

La începutul codului am definit câteva constante importante care controlează comportamentul algoritmului:

```
const int NUM_CLUSTERS = 5;              // Numărul de clustere k-means

const int MAX_IMAGES_PER_CLASS = 200;    // Max imagini per folder numeric

const int MIN_IMAGES_PER_CLASS = 20;     // Min imagini necesare per folder
```

```
const int VOCAB_SIZE = 100;              // Dimensiunea vocabularului BoW
const int MAX_FEATURES_PER_IMAGE = 100;   // Puncte SIFT per imagine


// Procesez doar aceste trei foldere numerice:
const set<string> SELECTED_CLASSES = { "00", "01", "02" };
```

Am ales să lucrez doar cu 3 foldere din dataset pentru a reduce timpul de procesare și a face testarea mai rapidă. În practică, se poate extinde la toate cele 15 clase.

Pasul 1: Încărcarea imaginilor

Funcția load3SceneNumeric() se ocupă de citirea imaginilor din folderele numerice. Am implementat această funcție să parcurgă directoarele și să identifice doar folderele din SELECTED_CLASSES:

```
for (const string& rawName : classDirs) {
   // Procesez doar dacă folderul este în SELECTED_CLASSES
   if (SELECTED_CLASSES.count(rawName) == 0) {
      continue;
   }
   cout << "Processing folder: " << rawName << " (ID: " << classId << ")\n";
   string classPath = datasetPath + "\\" + rawName;
   vector<string> imageFiles = getImageFiles(classPath);
```

Pentru fiecare imagine încărcată, fac un resize la 256×256 pixeli păstrând aspect ratio-ul:

```
// Redimensionez la 256×256 păstrând aspectul
int targetSize = 256;
double scale = (double)targetSize / max(img.rows, img.cols);
int newW = (int)(img.cols * scale);
int newH = (int)(img.rows * scale);


Mat resized;
resize(img, resized, Size(newW, newH), 0, 0, INTER_LINEAR);
```

```
// Adaug padding pentru a ajunge exact la 256×256
Mat padded = Mat::zeros(targetSize, targetSize, CV_8UC3);
int offsetX = (targetSize - newW) / 2;
int offsetY = (targetSize - newH) / 2;
resized.copyTo(padded(Rect(offsetX, offsetY, newW, newH)));
```

Pasul 2: Conversia la grayscale

Această etapă este simplă - SIFT funcționează pe imagini în grayscale, așa că convertesc toate imaginile:

```
void convertToGrayscale(const vector<Mat>& colorImgs, vector<Mat>& grayImgs) {
    grayImgs.clear();
    grayImgs.resize(colorImgs.size());

    for (size_t i = 0; i < colorImgs.size(); i++) {
        cvtColor(colorImgs[i], grayImgs[i], COLOR_BGR2GRAY);
    }
}
```

Pasul 3: Extragerea caracteristicilor SIFT

Aici folosesc algoritmul SIFT din OpenCV 4.x (nu mai am nevoie de xfeatures2d ca în versiunile mai vechi):

```
// Creez detectorul SIFT
detector = SIFT::create(MAX_FEATURES_PER_IMAGE);
// Pentru fiecare imagine, aplic preprocesare înainte de SIFT
Mat processed;
GaussianBlur(img, processed, Size(3, 3), 0.5);
Ptr<CLAHE> clahe = createCLAHE(2.0, Size(8, 8));
clahe->apply(processed, processed);
```

```
vector<KeyPoint> keypoints;
Mat desc;
detector->detectAndCompute(processed, noArray(), keypoints, desc);
```

Am adăugat și un fallback pentru imaginile care nu au suficiente keypoint-uri detectate automat:

```
if (desc.empty() || desc.rows == 0) {
    // Fallback: creez o grilă de keypoint-uri
    keypoints.clear();
    for (int y = 20; y < img.rows - 20; y += 40) {
        for (int x = 20; x < img.cols - 20; x += 40) {
            keypoints.emplace_back(Point2f(x, y), 16.0f);
        }
    }
    if (!keypoints.empty()) {
        detector->compute(processed, keypoints, desc);
    }
}
```

Pasul 4: Construirea vocabularului vizual

Pentru a construi vocabularul, colectez descriptori din toate imaginile și aplic k-means:

```
Mat buildVocabulary(const vector<Mat>& allDescriptors, int vocabSize) {
    Mat trainingData;
    RNG rng(12345);
    int maxSamples = 30000; // limitez numărul de sample-uri

    // Iau câte 20 de descriptori random din fiecare imagine
    for (const auto& desc : allDescriptors) {
```

```
        if (desc.empty()) continue;
        int sampleCount = min(desc.rows, 20);
        for (int i = 0; i < sampleCount; ++i) {
            if (trainingData.rows >= maxSamples) break;
            int idx = rng.uniform(0, desc.rows);
            trainingData.push_back(desc.row(idx));
        }
    }

    // Aplic k-means pe descriptori
    TermCriteria criteria(TermCriteria::MAX_ITER + TermCriteria::EPS, 80, 0.001);
    BOWKMeansTrainer bowTrainer(vocabSize, criteria, 3, KMEANS_PP_CENTERS);
    Mat vocabulary = bowTrainer.cluster(trainingDataFloat);

    return vocabulary;
}
```

Pasul 5: Calcularea histogramelor BoW

Pentru fiecare imagine, calculez histograma care arată cât de des apar cuvintele vizuale din vocabular:

```
// Creez extractor-ul BoW cu matcher-ul potrivit
Ptr<DescriptorMatcher> matcher = BFMatcher::create(NORM_L2);
Ptr<BOWImgDescriptorExtractor> bowExtractor =
    makePtr<BOWImgDescriptorExtractor>(detector, matcher);
bowExtractor->setVocabulary(vocabulary);

// Pentru fiecare imagine, calculez histograma
Mat bowHist;
bowExtractor->compute(processed, keypoints, bowHist);
```

```
// Normalizez histograma
normalize(bowHist, bowHist, 1.0, 0.0, NORM_L2);
```

Am adăugat verificări pentru cazurile în care histograma nu poate fi calculată:

```
if (bowHist.empty()) {
    bowHist = Mat::ones(1, vocabSize, CV_32F) * (1.0f / vocabSize);
} else {
    double normVal = cv::norm(bowHist);
    if (normVal == 0 || isnan(normVal) || isinf(normVal)) {
        bowHist = Mat::ones(1, vocabSize, CV_32F) * (1.0f / vocabSize);
    }
}
```

Pasul 6: Clustering-ul final cu k-means

După ce am toate histogramele BoW, aplic din nou k-means pentru a grupa imaginile similare:

```
vector<int> performKMeansClustering(const Mat& bowFeatures, int numClusters) {
    Mat labels, centers;
    TermCriteria criteria(TermCriteria::EPS + TermCriteria::MAX_ITER, 80, 1.0);

    double compactness = kmeans(bowFeatures, numClusters, labels,
                                criteria, 5, KMEANS_PP_CENTERS, centers);

    // Convertesc labels la vector<int>
    vector<int> clusterLabels((size_t)labels.rows);
    for (int i = 0; i < labels.rows; ++i) {
        clusterLabels[i] = labels.at<int>(i, 0);
    }
```

```
    return clusterLabels;
}
```

Pasul 7: Salvarea rezultatelor

În final, salvez imaginile organizate pe clustere și afișez niște statistici:

```
// Creez directoarele pentru fiecare cluster
for (int i = 0; i < NUM_CLUSTERS; ++i) {
    string clusterDir = outDir + "\\cluster_" + to_string(i);
    makeDir(clusterDir);
}

// Salvez fiecare imagine în clusterul corespunzător
for (size_t i = 0; i < images.size(); ++i) {
    int cluster = clusterLabels[i];
    int originalLabel = originalLabels[i];
    string originalClass = classNames[originalLabel];

    char buf[256];
    snprintf(buf, sizeof(buf), "img_%04zu_orig_%s.jpg", i, originalClass.c_str());
    string filepath = outDir + "\\cluster_" + to_string(cluster) + "\\" + buf;
    imwrite(filepath, images[i]);
}
```

Orchestrarea generală

Funcția main() și cluster3SceneNumeric() coordonează execuția tuturor pașilor în ordine. Am adăugat măsurarea timpului pentru a vedea cât durează procesul:

```
bool cluster3SceneNumeric(const string& datasetPath) {
    auto startTime = chrono::high_resolution_clock::now();
```

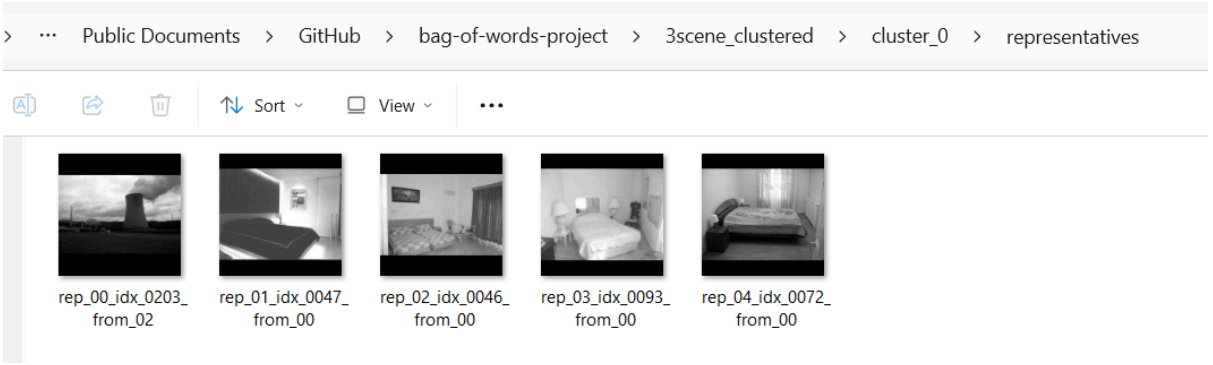
```
    // Execut toți cei 7 pași în ordine

    // ... (toți pașii)


    auto endTime = chrono::high_resolution_clock::now();

    auto duration = chrono::duration_cast<chrono::seconds>(endTime -
startTime).count();

    cout << "\n3-Scene BoW clustering completed in " << duration << " seconds\n";


    return true;

}
```

Această structură modulară face codul ușor de înțeles și de modificat. Fiecare funcție are o responsabilitate clară și poate fi testată separat. Am inclus și destule mesaje de debug pentru a urmări progresul și a identifica eventualele probleme.

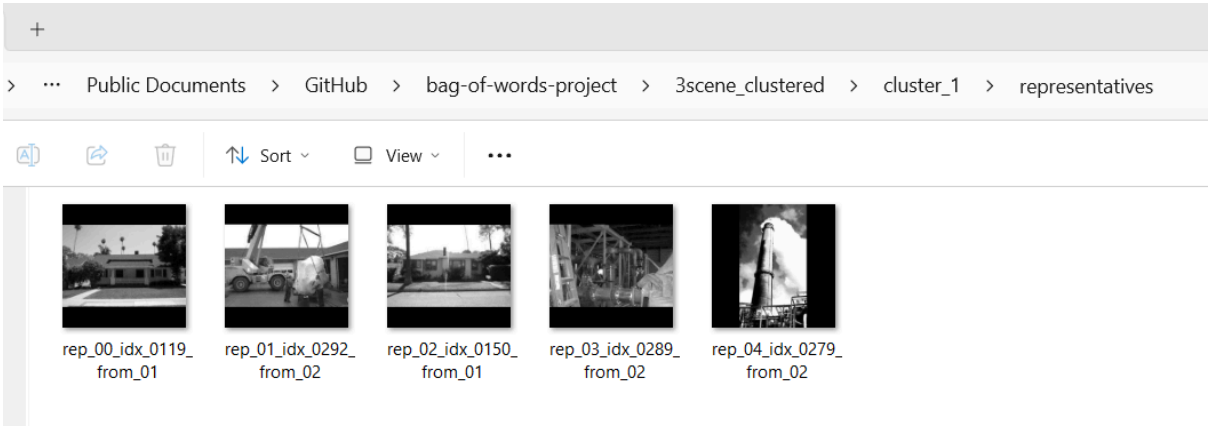
Rezultate

Rezultatele indică un rezultat de aproximativ 40-50% pentru 200 de cuvinte și 100 de poze pe clasa. E un rezultat promițător, iar dacă resursele hardware erau mai mari, ar fi crescut acest număr.

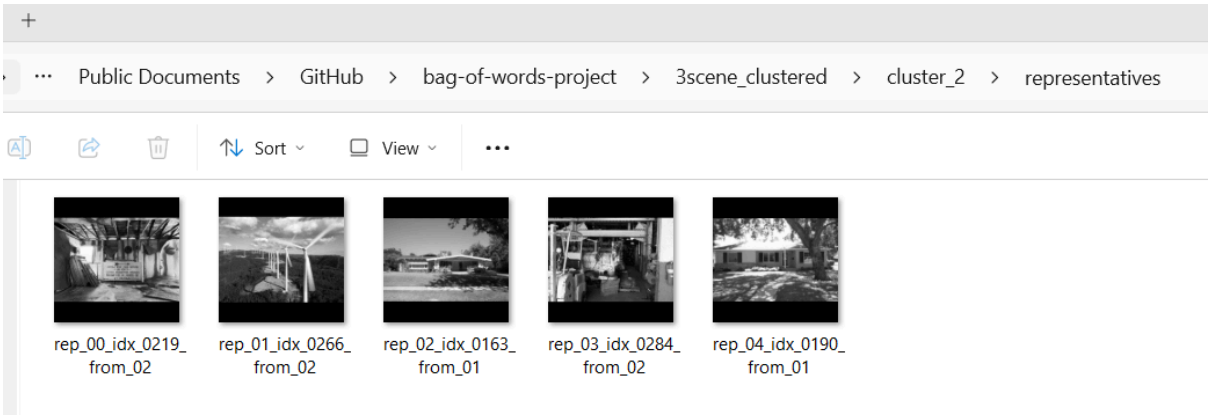
Reprezentanți cluster 1:



Reprezentanți cluster 2:



Reprezentanți cluster 3:



Rezultate finale după execuția programului:

```
=== 3-Scene BoW Clustering Analysis ===

Cluster Distribution:
  Cluster 0: 57 images
  Cluster 1: 110 images
  Cluster 2: 133 images

Cluster vs Original Folder Analysis:

Cluster 0 contains:
  00: 31 images
  02: 15 images
  01: 11 images

Cluster 1 contains:
  02: 45 images
  00: 34 images
  01: 31 images

Cluster 2 contains:
  01: 58 images
  02: 40 images
  00: 35 images
  Saved 5 representatives for cluster 0
  Saved 5 representatives for cluster 1
  Saved 5 representatives for cluster 2
```

Bibliografie

D. Lowe, Distinctive Image Features from Scale-Invariant Keypoints, International Journal of Computer Vision, vol. 60, no. 2, pp. 91–110, 2004.

A. Coates, H. Lee, and A. Ng, "An Analysis of Single-Layer Networks in Unsupervised Feature Learning," in Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS), 2011 – STL-10 Dataset Description.

J. Sivic and A. Zisserman, "Video Google: A Text Retrieval Approach to Object Matching in Videos," in Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2003.

G. Csurka, C. Dance, L. Fan, J. Willamowski, and C. Bray, "Visual Categorization with Bags of Keypoints," in Workshop on Statistical Learning in Computer Vision, ECCV, 2004.

T. Tuytelaars and K. Mikolajczyk, "Local Invariant Feature Detectors: A Survey," Foundations and Trends® in Computer Graphics and Vision, vol. 3, no. 3, pp. 177–280, 2008.

H. Bay, T. Tuytelaars, and L. Van Gool, "SURF: Speeded-Up Robust Features," in Proceedings of the European Conference on Computer Vision (ECCV), 2006.

E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An Efficient Alternative to SIFT or SURF," in Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2011.

R. Datta, D. Joshi, J. Li, and J. Z. Wang, "Image Retrieval: Ideas, Influences, and Trends of the New Age," ACM Computing Surveys (CSUR), vol. 40, no. 2, pp. 1–60, 2008.

R. Szeliski, Computer Vision: Algorithms and Applications, Springer, 2010.

OpenCV Documentation – Feature Detection and Description Modules: https://docs.opencv.org.

OpenCV Bag of Words tutorial by Roy Shilkrot: https://github.com/shilkrot/BoWImageClassifier.

K. Grauman and B. Leibe, Visual Object Recognition, Morgan & Claypool, 2011.

K. Mikolajczyk and C. Schmid, "A Performance Evaluation of Local Descriptors," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 27, no. 10, pp. 1615–1630, 2005.

FLANN – Fast Library for Approximate Nearest Neighbors: https://www.cs.ubc.ca/research/flann/.

C. Sanderson and B. C. Lovell, "Multi-Region Probabilistic Histograms for Robust and Scalable Identity Inference," in Lecture Notes in Computer Science, vol. 5558, pp. 199–208, 2009.