COMP3204 CourseWork

Zhengbin Lu
33839239

Abstract

This project investigates scene recognition through three distinct methodologies, showcasing the evolution from traditional to advanced techniques in computer vision. The first approach utilizes a k-nearest-neighbor classifier with tiny image features, emphasizing simplicity. The second employs a bag-of-visual-words representation with linear support vector machines, highlighting feature encoding strategies. The third leverages a deep learning paradigm with transfer learning using a pre-trained ResNet-50 model, illustrating state-of-the-art capabilities. Experimental results demonstrate the comparative effectiveness of these approaches, providing insights into their strengths and limitations in addressing the scene classification challenge.

1. Introduction

This project explores scene recognition, a key task in computer vision, by implementing and evaluating three classification methods. The first uses "tiny image" features with a k-nearest-neighbour classifier, the second employs a bag-of-visual-words model with linear classifiers, and the third investigates advanced techniques to optimize recognition performance.

2. Run 1

2.1. Description

This experiment aims to classify grayscale images into 15 distinct scene categories. In Run1, a k-Nearest Neighbor (KNN) classifier was employed using the "tiny image" feature representation. Each image was cropped to a central square, resized to a fixed dimension of 16×16 pixels, and normalized to ensure zero mean and unit variance. This process aimed to standardize the features and reduce the influence of brightness variations. The KNN classifier was trained on these features, with the optimal hyperparameters determined using 10-fold cross-validation. The trained model was subsequently applied to classify the test set, and the predictions were saved in the required format.

2.2. Implementation

The feature extraction process consisted of three main steps: cropping, resizing, and normalization. Each image was first cropped to a central square region based on a configurable side length, effectively removing irrelevant information from the image boundaries. The cropped region was then resized to a uniform dimension of 16×16 pixels, ensuring consistent feature representation across all images. Subsequently, pixel values were flattened into a one-dimensional vector and normalized. The normalization process involved adjusting pixel values to have a zero mean and scaling them to unit variance, reducing the influence of brightness variations and ensuring stability and consistency in distance calculations during classification.

During the data loading phase, images were recursively processed according to their categories. For each valid image, its features were extracted and stored in a feature matrix, where each row represented the feature vector of a single image. The corresponding class labels were encoded as integer indices to enhance processing efficiency. This setup ensured orderly feature extraction and storage while maintaining compatibility with the classification algorithm.

Given the high dimensionality of the flattened feature vectors (256 dimensions, corresponding to 16×16), Principal Component Analysis (PCA) was employed for dimensionality reduction. PCA projected the data onto its principal components, preserving the most significant variance in the data while effectively filtering out noise. The reduced features not only substantially decreased computational complexity but also provided a more concise and meaningful representation for model training and testing.

During model training, the KNN classifier employed a distance-based weighting strategy, where closer neighbors contributed more significantly to the classification decision. The distance metric used was the Euclidean distance. The training phase utilized the reduced feature set and its corresponding labels, ensuring both the accuracy of classification and the computational efficiency of the model.

2.3. Training and Tuning

During the training process, three key parameters were carefully tuned to optimize performance. The first param-

eter was the number of neighbors (k) in the KNN classifier. The range for k was set to $[1, 3, 5, \ldots, 17]$, with only odd values considered to avoid tie scenarios during classification. Through cross-validation, the optimal value of $k = 11$ was identified, ensuring a good balance between model complexity and classification accuracy.

The second parameter involved the number of principal components used for dimensionality reduction. The range for this parameter was set between 20 and 60 components, allowing the model to retain significant features while reducing computational complexity. Experimental results indicated that 35 components provided the best trade-off, effectively preserving critical information without unnecessary overhead.

The third parameter was the side length of the central cropping region during feature extraction. This side length was varied between 200 and 600 pixels to determine its impact on model performance. A side length of 400 pixels was found to be optimal, as it retained sufficient image content for effective feature representation while avoiding the inclusion of irrelevant regions caused by excessive cropping.

2.4. Optimal hyper parameters and accuracy

The optimal configuration for the Run1 model included $k = 11$, 35 principal components for PCA, and a cropping side length of 400 pixels. Under these settings, the model achieved a 10-fold cross-validation accuracy of 27.1%, representing the best performance achievable under the given conditions.

2.5. Prediction

In Run1, the "tiny image" features and KNN classifier were successfully utilized for scene classification. During testing, the preprocessing pipeline was applied consistently to the test data. The model, configured with the optimal parameters, classified the test images, and the results were stored in the required format in `run1.txt`. To organize the prediction results, the system employed a natural sorting algorithm, which extracts the numeric parts of filenames using regular expressions to achieve a natural order.

3. Run2

3.1. Description

Run2 focuses on performing scene classification using Bag-of-Visual-Words (BoVW) features combined with a set of linear classifiers (One-vs-All SVM). The implementation involves extracting fixed-size pixel patches from grayscale images, generating a visual vocabulary through K-Means clustering, encoding each image as a histogram feature vector, and training a set of linear Support Vector Machine (SVM) classifiers on these features. Key parameters were tuned to optimize classification performance, and the trained model was applied to the test set to generate classification predictions.

3.2. Implementation

The system employs parallelization to efficiently load training and testing data. By leveraging the 'joblib' library's 'Parallel' and 'delayed' functions, the system processes image reading and patch extraction in a multi-threaded environment. This parallelized approach significantly reduces data loading time, especially when handling large-scale image datasets.

During the preprocessing phase, the system adopts a sliding window method to extract fixed-size patches from images, with a patch size of 8×8 pixels and a stride of 4 pixels. This dense sampling strategy ensures comprehensive coverage of different regions of the image. To enhance feature robustness, each extracted patch undergoes mean-centering and standard deviation normalization. This process subtracts the patch's mean and divides by its standard deviation, effectively mitigating the impact of illumination variations and ensuring consistency in feature distribution across patches.

In the Bag-of-Visual-Words (BoVW) model, feature clustering constitutes a critical step. The system employs the standard KMeans algorithm to cluster the features, generating a visual vocabulary initialized with 500 clusters. Despite its higher computational complexity, KMeans produces precise and stable cluster centers that effectively represent image features. To ensure balanced feature representation across categories, the system adopts a stratified sampling strategy, selecting a fixed number of patches from each class. This approach mitigates clustering bias caused by class imbalance while balancing computational efficiency and clustering quality.

After clustering, the system performs vector quantization, mapping each image patch to its nearest cluster center. The frequency of each visual word in the image is then computed to construct the BoVW histogram. To handle large-scale data efficiently, the histogram is converted into a sparse matrix in Compressed Sparse Row (CSR) format. This sparse representation significantly reduces memory usage and computational overhead, particularly in high-dimensional feature vectors where most elements are zero.

Following the generation of sparse histograms, the system standardizes features using the 'StandardScaler' utility. Given the sparse matrix format, the standardization process omits mean subtraction ('with_mean=False') to preserve sparsity. This ensures that each feature dimension has zero mean and unit variance, improving the stability and performance of classifier training.

To reduce the complexity of high-dimensional features, the system employs Truncated Singular Value Decomposition (TruncatedSVD). Unlike traditional Principal Com-

ponent Analysis (PCA), TruncatedSVD directly handles sparse matrices, reducing the feature dimensionality to 45 while retaining the data's primary structural information. This dimensionality reduction enhances computational efficiency and reduces model complexity.

For classification, the system utilizes a One-vs-All Support Vector Machine (SVM) classifier architecture, where each binary SVM corresponds to a specific class. The choice of a linear kernel ensures high computational efficiency and strong classification performance in high-dimensional feature spaces. To accelerate the training process, the system applies 'joblib'-based parallelization, leveraging multi-core CPU resources to speed up the training of multiple classifiers.

During evaluation, the system adopts stratified K-fold cross-validation to ensure that the class distribution in each fold matches that of the overall dataset. This strategy reduces bias from imbalanced data distributions, providing a more reliable assessment of model performance. By recording the accuracy of each fold and computing the mean accuracy, the system offers a comprehensive evaluation of the classifier's generalization capabilities.

To optimize runtime performance, the system extensively employs parallelization techniques to accelerate key steps such as data loading, feature extraction, classifier training, and prediction. Additionally, memory management is carefully handled by releasing unused variables and invoking garbage collection ('gc.collect()') at critical stages. This approach prevents memory leaks and excessive memory consumption, ensuring the system's stability and scalability, especially when handling high-dimensional sparse data and large numbers of patches.

3.3. Training and Tuning

The training process systematically explored and optimized key parameters to achieve the best configuration. Regarding patch size and stride, various patch sizes ranging from 2×2 to 10×10 were evaluated, with 5×5 being identified as the optimal size. Smaller patches, such as 2×2, focused excessively on localized features, resulting in a lack of global information, while larger patches, such as 10×10, were insufficient in capturing fine details, particularly in texture-rich regions. For stride, a range of 2 to 8 pixels was tested, and a stride of 4 pixels was chosen to balance computational efficiency and feature coverage.

In terms of vocabulary size, a range of 500 to 1000 was assessed, with 700 determined as the optimal value. Smaller vocabularies, such as 500, were inadequate for representing local patterns, whereas larger vocabularies, such as 1000, increased both sparsity and computational cost. Furthermore, for Principal Component Analysis (PCA), components between 30 and 60 were explored, with 45 components achieving the best trade-off between feature retention and computational efficiency.

Finally, in the selection of training samples for K-Means clustering, the range of sampled patches varied from 5000 to 16500. Ultimately, 15000 samples were identified as sufficient to represent local patterns comprehensively while maintaining computational feasibility.

3.4. Optimal hyper parameters and accuracy

The optimal configuration for Run2 was determined with a patch size of 5×5, a stride of 4 pixels, a vocabulary size of 700, 45 components, and 15,000 clustering samples. Using this configuration, the model achieved its highest performance during 10-fold cross-validation, with a validation accuracy of 68 $\%$.

3.5. Prediction

During the prediction phase, the model trained with the optimal hyperparameter combination (sample size: 15,000, number of components: 45, stride: 4, window size: 5×5) was used to evaluate the test set. To organize the prediction results, the system employed a natural sorting algorithm, which extracts the numeric parts of filenames using regular expressions to achieve a natural order.

4. Run3

4.1. Description

Run3 aims to design and implement a high-performance image classifier by combining advanced feature extraction techniques with sophisticated classification methods. In this experiment, we utilized a transfer learning approach based on the ResNet-50 pre-trained model to address the scene classification task. The methodology encompassed data preprocessing, model design and fine-tuning, training and validation, and final test set prediction with result storage.

4.2. Implementation

The classifier employs various data augmentation techniques to significantly enhance its generalization capability in data processing. For the training set, grayscale images are first converted to three-channel images by replicating the single channel, ensuring compatibility with pretrained models that typically require three-channel input. Subsequently, random cropping and resizing are applied to standardize the image size to 224x224 pixels. Random cropping not only enables the model to focus on different parts of the image but also enhances data diversity by altering perspectives. Additionally, random horizontal flipping and rotation operations further increase the variability of the training data, enabling the model to maintain high recognition accuracy across images with different orientations and angles. Color jittering is also employed to randomly adjust the brightness, contrast, saturation, and hue of the images, compelling the

model to learn more robust feature representations and mitigating performance fluctuations caused by lighting variations. Finally, the images are converted into PyTorch tensors and standardized by subtracting the mean and dividing by the standard deviation, scaling the pixel values to the range of [-1, 1].

For the validation and test sets, a more conservative preprocessing approach is adopted to ensure that the data distribution during evaluation and prediction aligns with real-world scenarios. This involves resizing the shorter edge of the image to 256 pixels and performing a center crop to obtain a 224x224-pixel image region. This approach avoids the distribution bias introduced by random transformations and ensures the stability and reliability of model evaluation and prediction. The normalization procedure remains consistent with the training set to guarantee uniformity in data input.

Regarding data loading, a highly customized Dataset class is implemented to efficiently manage and process image data. During image loading, exceptions are handled by generating blank images, enhancing the robustness of the data loading process.

In terms of model training, the classifier adopts the pretrained ResNet-101 model and adjusts its fully connected classification layer according to specific task requirements to accommodate different numbers of categories. ResNet-101, a deep residual network, offers a deep architecture and strong feature representation capabilities, making it well-suited for handling complex classification tasks. Leveraging a pretrained model allows the classifier to utilize the rich feature representations learned from large-scale datasets, significantly accelerating the convergence of the training process and improving initial model performance.

During training, mixed-precision training techniques are employed, utilizing GradScaler and autocast to achieve mixed computation with float16 (FP16) and float32 (FP32). This technique not only significantly improves training efficiency but also reduces memory usage. The classifier also employs a cosine annealing learning rate scheduler to gradually reduce the learning rate, facilitating more stable convergence to a global optimum in the later stages of training. The cosine annealing scheduler dynamically adjusts the learning rate during training, maintaining a high learning rate in the early stages to rapidly explore the parameter space and gradually lowering it in the later stages to refine parameter adjustments and avoid local optima. The periodic adjustment of the learning rate further helps the model escape local optima, thereby enhancing overall performance.

For the loss function, the classifier uses Cross-Entropy Loss, a widely used loss function in multi-class classification tasks that effectively measures the difference between the predicted probability distribution and the true labels. The Adam optimizer is employed, which is an adaptive learning rate optimization algorithm based on first-order gradients, allowing for efficient navigation of complex loss surfaces to find optimal solutions. To prevent overfitting, an early stopping mechanism is introduced during training to avoid excessive fitting on the training set. Additionally, the classifier adopts multithreaded data loading (by configuring multiple worker processes) and memory pinning (pin memory=True) strategies during data loading, further enhancing the speed and efficiency of the data loading process.

4.3. Training and Tuning

The training process involved systematic exploration and tuning of key hyperparameters. The batch size, ranging from 16 to 64, was finalized at 32 to balance computational efficiency and convergence stability. The initial learning rate, explored between 1×10^{-5} and 1×10^{-3}, was set to 1×10^{-4}, achieving optimal convergence speed while avoiding instability. The learning rate scheduler reduced the learning rate every 7 epochs, a strategy informed by the observed performance improvement curves. The maximum number of training epochs was set to 25, with an early stopping criterion triggered by no improvement in validation accuracy for 5 consecutive epochs.

Data augmentation, including random cropping, horizontal flipping, and rotations within ±20°, improved the model's ability to generalize. These augmentations complemented standardization using ImageNet-specific mean and standard deviation values, aligning the input data with the distribution expected by ResNet-50. The use of transfer learning further simplified training by freezing the pretrained convolutional layers, allowing the newly initialized linear layer to adapt efficiently to the classification task.

4.4. Optimal hyper parameters and accuracy

The optimal model configuration included a batch size of 32, an initial learning rate of 1×10^{-4} with decay every 7 epochs, and a maximum of 25 epochs with early stopping. Data augmentation and transfer learning strategies further enhanced performance. Under these settings, the model achieved a validation accuracy of 94%.

4.5. Prediction

The test images underwent standardized preprocessing to ensure consistency with the training pipeline. Using the best-performing weights from the validation set, the model predicted class probabilities for each test image and selected the class with the highest probability as the classification result. These predictions were saved in the required format as `run3.txt`. To organize the prediction results, the system employed a natural sorting algorithm, which extracts the numeric parts of filenames using regular expressions to achieve a natural order.