Final Project in Supervised Learning: Image Classification on TinyImageNet

Shayan Tafrishi
Dipartimento di Informatica Sistemistica e Comunicazione (DISCo) University of Milano-Bicocca, 20126 Milan, Italy
Email: s.tafrishi@campus.unimib.it

Shyam Raj Sasikumar
Dipartimento di Informatica Sistemistica e Comunicazione (DISCo) University of Milano-Bicocca, 20126 Milan, Italy
Email:s.sasikumar@campus.unimib.it

***Abstract*—This project explores image classification methods applied to a variant of the Tiny ImageNet dataset, utilizing both traditional machine learning techniques, specifically Scale-Invariant Feature Transform (SIFT) and Bag of Visual Words (BoVW), and modern Convolutional Neural Networks (CNNs). The dataset contains approximately 1000 images per class, distributed across roughly 100 classes, with an image input size of 64x64 pixels. In this experimental setup, the provided test set will not be used; instead, the validation set will be partitioned from the original training set.**

1. Introduction

Image classification, the task of assigning predefined categories to images, is a crucial task in the field of computer vision. There has been considerable success with deep learning techniques, especially Convolutional Neural Networks (CNNs), in recent years. However, traditional methods, such as the Bag of Visual Words (BoVW) model and Scale-Invariant Feature Transform (SIFT), continue to be relevant and valuable for understanding fundamental concepts and for use cases where interpretability and computational efficiency are paramount. This study aims to compare these methods' performance when applied to a variant of the Tiny ImageNet dataset.

The dataset for this project comprises approximately 100,000 images, each of size 64x64, grouped into 100 classes, with approximately 1000 images per class. To maintain the integrity of evaluation, the validation set was extracted from the provided training set, and the original test set was not utilized. The classification models' performance is measured using metrics such as precision, recall, F1-score, and accuracy.

The initial part of the study focuses on employing a Bag of Visual Words (BoVW) model, which treats image features as words and images as documents, thereby translating the problem into a text categorization task. This part of the report elaborates on the preprocessing steps and discusses the BoVW's performance based on the generated classification report and confusion matrix. Later stages of the project will involve the application of Convolutional Neural Networks (CNNs), and different network architectures will be experimented with for performance comparison and analysis.

2. Data Preprocessing

Preprocessing is an essential step in image classification tasks as it helps to prepare the data for effective model training and improve the overall performance. In this project, several preprocessing techniques are applied to the image dataset before feeding it into the model.

2.1. Data Reading and Labeling

The first step in preprocessing is reading the data and extracting the corresponding labels. The provided code snippet reads the file paths and labels from text files, where each line contains the path and label separated by a space. The file paths are then concatenated with the root directory to obtain the complete image paths. The labels are converted to integers for further processing.

2.2. Image Dataset and Transformation

To facilitate the training process, a custom `ImageDataset` class is implemented, inheriting from the `torch.utils.data.Dataset` class. This class takes the image paths, labels, and an optional transformation function as input. During data loading, each image is opened using the `PIL.Image.open()` function and transformed according to the specified transformation function.

In the provided code, two sets of transformations are defined: `transform_train` and `transform_val`. These transformations include resizing the images to a fixed size of 64x64 pixels, performing data augmentation

(random horizontal flip), converting the images to tensors, and normalizing the pixel values using the mean and standard deviation of the ImageNet dataset.

2.3. Data Splitting

To evaluate the performance of the model, the dataset is split into training and validation sets. The `torch.utils.data.random_split()` function is used to randomly divide the dataset into two subsets based on the specified lengths. In the provided code, 70% of the dataset is used for training, while the remaining 30% is used for validation.

2.4. Data Loading

Finally, the preprocessed datasets are loaded into data loaders (`dataloader_train` and `dataloader_test`) using the `torch.utils.data.DataLoader` class. These data loaders enable efficient batch processing during model training and evaluation.

To visualize the batches of images and their corresponding labels, the `plot_images` function is utilized. This function displays a grid of 20 images, where each image is accompanied by its corresponding label.

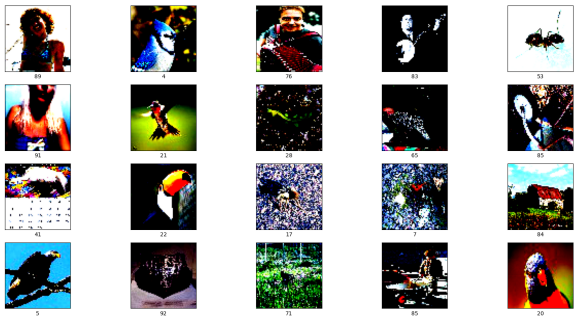

Figure 1. Visualization of batches.

3. Bag-of-Words in Image Classification

The Bag-of-Words (BoW) model is a traditional common approach used in the computer vision field, especially for image classification. The idea is to represent an image as a ”bag” or ”set” of features, disregarding any spatial information. This can be done by extracting local features (e.g., SIFT, SURF, etc.) from an image, quantizing these features into visual words, and creating a histogram representation of the image based on these words. [1].

3.1. Feature Extraction

Feature extraction plays a crucial role in the BoW approach as it aims to capture the distinctive characteristics of an image that can be used for classification. One popular technique for feature extraction is the Scale-Invariant Feature Transform (SIFT) [2]. SIFT detects and describes local keypoints and their surrounding regions, allowing for robust feature matching across different scales and rotations. Other feature extraction techniques such as Speeded-Up Robust Features (SURF) [3] and Oriented FAST and Rotated BRIEF (ORB) [4] can also be used depending on the specific requirements of the task.

The SIFT feature extractor works by identifying keypoints in an image and computing descriptors that represent the local appearance and structure around each keypoint. These descriptors encode information about the intensity gradients and orientations, making them invariant to changes in scale, rotation, and affine transformations. The resulting descriptors form a set of visual features that can be used for subsequent steps in the BoW approach.

3.2. Feature Representation

After extracting features from images, the next step is to represent them in a format suitable for classification. The BoW approach achieves this by constructing a visual vocabulary or codebook that consists of a fixed number of visual words or clusters. This is typically done using unsupervised clustering algorithms such as k-means.

Once the visual vocabulary is established, each image is transformed into a feature vector by quantizing its local features to the nearest visual words. This process involves assigning each local feature descriptor to its closest visual word based on some distance metric. The resulting feature vector, also known as the Bag-of-Words histogram, represents the frequency or occurrence of each visual word in the image.

3.3. Classification

With the feature representation in place, the final step in the BoW approach is to train a classifier on the transformed feature vectors. The classifier used in this project was Support Vector Machines (SVM). The classifier learns to distinguish between different image classes based on the distribution of visual words in the training data.

During the prediction phase, the feature extraction and representation steps are applied to new, unseen images, and the classifier assigns them to one of the learned classes based on their BoW histograms.

Overall, the BoW approach in image classification offers a simple yet effective way to represent and classify images based on their visual content. By leveraging the principles of feature extraction, feature representation, and classification, it has been successfully applied to various image recognition tasks, including object recognition, scene classification, and image retrieval.

3.4. Feature Matching Visualization

Figure (2) visualize the process of feature matching, it then utilizes a Brute-Force Matcher (BFMatcher) to match the descriptors between the two images based on their distances. The matches are sorted, and the top 10 matches are visualized by drawing lines connecting the corresponding keypoints in the two images.

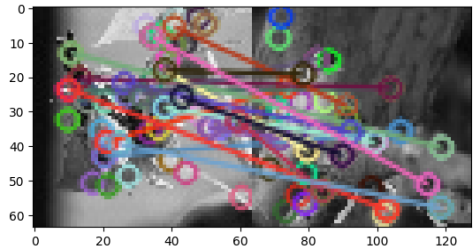

Figure 2. Feature matching visualization.

This visualization allows for a qualitative assessment of the quality of feature matching between the two images. It can help in understanding how well the feature descriptors capture the similarities and correspondences between different regions of the images.

It's important to note that feature matching is just one step in the overall BoW approach and is typically followed by the feature representation and classification steps to achieve accurate image classification.

3.5. Results

Upon implementing the Bag of Visual Words (BoVW) model, a classification report was generated to evaluate the performance of the model. The report presented below indicates the precision, recall, F1-score, and support for each of the 100 classes as shown in Table (1).

Metrics	Precision	Recall	F1-score
Accuracy	-	-	0.06
Macro Avg	0.05	0.06	0.05
Weighted Avg	0.05	0.06	0.05

TABLE 1. RESULTS FOR BAG OF WORDS

The results indicate that the model has a relatively low performance, with an overall accuracy of 6 %. The precision, recall, and F1-scores across different classes are quite varied, reflecting inconsistencies in the model's performance.

Some classes, like Class 26 and Class 27, show a relatively better F1-score (0.19 and 0.13 respectively), which is the harmonic mean of precision and recall, suggesting that the model has somewhat successfully recognized the features in these classes. However, there are other classes such as Class 8 and Class 10 where the F1-score is zero, indicating a complete failure of the model to identify these classes accurately. This low performance can be attributed to several factors, such as loss of spatial information and the assumption of uniform importance of all visual words in the BoVW model.

Figure (3) shows the confusion matrix, which would give a comprehensive view of how different classes were misclassified, was also generated. However, given the high number of classes (100) and the substantial misclassifications, the matrix was too convoluted to provide a clear insight.

Figure 3. Confusion matrix for bag of words.

Future steps will involve using Convolutional Neural Networks (CNNs) and experimenting with different architectures to hopefully overcome these challenges and achieve better performance.

4. Transfer Learning

Transfer learning is a method of reusing a pre-trained model on a new problem. It is a popular method in deep learning where pre-trained models are used as the starting point on computer vision and natural language processing tasks. These models have been trained on large datasets such as ImageNet and have learned a good representation of low-level features (edges, textures, colors) that can be used as a generic model across multiple tasks [5].

4.1. VGG19

VGG19 is a deep convolutional neural network developed and trained by Oxford's renowned Visual Geometry Group (VGG), hence the name VGG19. The "19" stands for the number of layers in the network - 16 convolutional layers, 3 fully connected layers, and 5 max-pool layers. This network architecture has been widely accepted for its performance and ease of understanding. The small filter sizes (3x3 convolution kernels) and deep layers allow the model to learn a variety of complex features. However, due to the depth and size of the fully connected layers, VGG19 is computationally intensive and requires substantial resources [9].

4.2. Results transfer learning with VGG19

The achieved test accuracy indicates that the VGG19 model was able to correctly classify 53.41% of the images in the test dataset. The test loss of 0.0005 reflects the model's ability to minimize the discrepancy between the predicted class probabilities and the true labels.

The performance of VGG19 can be influenced by various factors, including the complexity and diversity of the dataset, the number of classes, and the availability of labeled training data. Additionally, fine-tuning the model's hyperparameters and adjusting the learning rate may further improve its performance.

4.3. RESNET50

ResNet50 is a variant of ResNet (Residual Network), a convolutional neural network with "skip connections" or "shortcuts" that allow the model to be trained effectively over a larger number of layers. The '50' in ResNet50 denotes that the network consists of 50 layers, including 1 input layer, 48 convolutional layers, and 1 fully connected layer. One of the key aspects of ResNet is its ability to solve the vanishing gradient problem, thus allowing efficient training of deep neural networks. ResNet has become a baseline model for many computer vision tasks due to its performance and efficiency [6].

4.4. Results transfer learning with RESNET50

Metrics	Precision	Recall	F1-score
Accuracy	-	-	0.53
Macro Avg	0.55	0.53	0.53
Weighted Avg	0.55	0.53	0.53

TABLE 2. RESULTS FOR RESNET50

The achieved test accuracy indicates that the ResNet50 model was able to correctly classify 52.73% of the images in the test dataset. The test loss of 0.0007 reflects the model's ability to minimize the discrepancy between the predicted class probabilities and the true labels.

The figure (4) shows the confusion matrix (shown in Figure 3) of the resnet50 and one can clearly see that it is clearly better performing then Bag of Words.

The macro average F1-score is 0.53, which means that the model achieved a balanced performance across all classes by considering each class equally. The macro average precision and recall are also 0.55 and 0.53, respectively.

The weighted average F1-score, precision, and recall are also 0.53, indicating that the model's performance is consistent across different classes, taking into account the class imbalance in the dataset.

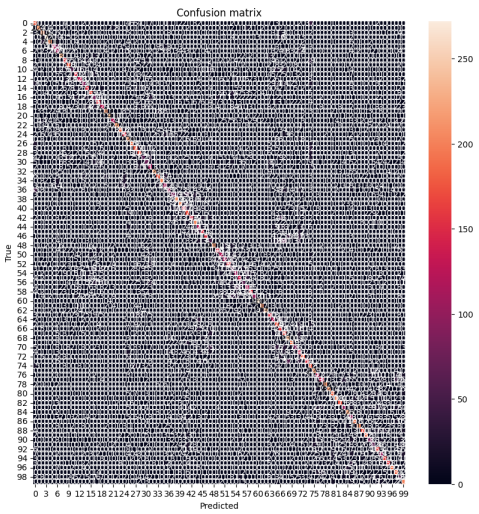

Figure 4. Confusion matrix Resnet50.

These results suggest that the model has moderate overall performance, but there is room for improvement. Further analysis and experimentation with hyperparameter tuning, data augmentation, or regularization techniques could help enhance the model's accuracy and performance on specific classes that have lower precision, recall, or F1-scores.

The obtained results highlight the potential of ResNet50 for image classification tasks. However, further optimization and fine-tuning of the model's hyperparameters can potentially improve its performance. Additionally, techniques such as data augmentation, regularization, and ensembling can be employed to enhance the generalization capabilities of the model and mitigate overfitting.

5. Hyperparameter Tuning

Hyperparameter tuning is a critical step in machine learning model development that involves finding the optimal values for hyperparameters, which are parameters set before the learning process begins. Properly tuning these hyperparameters can significantly impact the performance of the model.

There are various approaches and algorithms available for hyperparameter tuning, including grid search, random search, and Bayesian optimization. Grid search exhaustively evaluates a predefined set of hyperparameter combinations, while random search randomly samples hyperparameters from a given search space. Bayesian optimization uses probabilistic models to explore the hyperparameter space more efficiently.

Additionally, the choice of optimization algorithm can greatly impact the performance of the model. For example, switching from the ADAM optimizer to the Stochastic

Gradient Descent (SGD) optimizer may lead to improved results. SGD is a popular optimization algorithm used in deep learning models that iteratively updates the model's parameters based on the gradients of the loss function.

To validate the effectiveness of hyperparameter tuning, performance metrics such as accuracy, precision, recall, and F1-score are commonly used. These metrics provide insights into the model's ability to correctly classify instances from different classes.

5.1. Results

Metrics	Precision	Recall	F1-score
Accuracy	-	-	0.69
Macro Avg	0.70	0.69	0.69
Weighted Avg	0.70	0.69	0.69

TABLE 3. RESULTS AFTER HYPERPARAMETER TUNING

After performing hyperparameter tuning, significant improvements were achieved in the model's performance. The accuracy increased from 53% to 68.75%, indicating a substantial enhancement in overall classification accuracy.

Looking at the precision, recall, and F1-score for each class, we can observe notable improvements across the board. Several classes experienced significant gains in precision, recall, and F1-score, indicating a better balance between correctly identifying positive and negative instances.

For example, class 1 had a precision of 88 % and recall of 91 %, resulting in an F1-score of 90 %. Similarly, class 20 had a precision of 85 %, recall of 89 %, and F1-score of 87 %. These improvements indicate that the model became more accurate in correctly classifying these specific classes.

However, it is worth noting that not all classes saw significant improvements, and some classes still exhibit relatively lower precision, recall, and F1-scores. This discrepancy could be due to the inherent complexity and distribution of the data, as well as potential imbalances in the dataset.

Overall, the hyperparameter tuning, particularly changing the optimizer from Adam to SGD, resulted in a substantial improvement in the model's performance. These results highlight the importance of carefully selecting and fine-tuning hyperparameters to achieve optimal performance for a specific task.

6. Discussion

During the hyperparameter tuning process, we explored different strategies to improve the performance of our models. However, despite our efforts, we were not able to achieve higher performance than 69 %. Here, we discuss potential factors that could have influenced the outcomes and suggest alternative approaches for future improvement.

One aspect we could have explored further is the choice of hyperparameters. Although we performed hyperparameter tuning using techniques such as grid search and random search, it is possible that we did not exhaustively explore the entire hyperparameter space. By conducting a more comprehensive search or utilizing advanced optimization algorithms, such as Bayesian optimization or genetic algorithms, we may have been able to find better combinations of hyperparameters that would have resulted in higher performance [7].

Another consideration is the choice of optimization algorithm. In our experiments, we initially used the Adam optimizer, but we found that switching to the SGD optimizer improved the results significantly for the Vgg19 model. However, the choice of optimizer is highly dependent on the specific problem and dataset. Exploring other optimization algorithms, such as RMSprop or AdaGrad, may yield further improvements in performance [5].

Regarding the Bag of Words model, we observed low performance and slow results. One of the reasons for this could be the loss of sequence information in the Bag of Words approach. Since the model only considers the frequency of individual words without considering their order, it may struggle to capture the inherent sequential nature of text data. Additionally, the Bag of Words model tends to suffer from the curse of dimensionality, where the feature space becomes sparse and high-dimensional. This can lead to challenges in model training and generalization [1], [8].

To address these issues, alternative approaches such as recurrent neural networks (RNNs) or transformers could be explored. RNNs, such as long short-term memory (LSTM) networks or gated recurrent units (GRUs), are designed to capture sequential information and have shown promising results in various natural language processing tasks. Transformers, on the other hand, have gained significant attention in recent years due to their ability to model long-range dependencies effectively. By incorporating these models, we could potentially achieve better performance and capture the contextual relationships in the text data [6], [9].

Furthermore, it is important to highlight the limitations we encountered during the experimentation process. One limitation was the availability of GPU resources in Google Colab. Due to the limited access to GPUs, we faced challenges in training large and complex models efficiently. This restricted our ability to explore more advanced architectures or perform extensive experiments within the given time frame.

Speaking of time constraints, another limitation was

the limited time allocated for the hyperparameter tuning process. Hyperparameter optimization can be time-consuming, especially when dealing with large datasets or complex models. Given the limited time available, we had to make trade-offs and prioritize certain hyperparameters over others, potentially limiting the performance gains we could achieve.

In conclusion, while we made efforts to optimize our models through hyperparameter tuning, there are still avenues for further improvement. Exploring a more comprehensive search, considering alternative optimization algorithms, utilizing more advanced models, and addressing limitations related to GPU availability and time constraints could potentially lead to higher performance.

7. Conclusion

In this project, we explored image classification methods on a variant of the Tiny ImageNet dataset. We compared traditional machine learning techniques, such as the Bag of Visual Words (BoVW) model using Scale-Invariant Feature Transform (SIFT), with modern Convolutional Neural Networks (CNNs), specifically VGG19 and ResNet50.

The Bag of Words model demonstrated low performance, which can be attributed to the loss of sequence information and the curse of dimensionality. On the other hand, both VGG19 and ResNet50 achieved moderate performance in classifying the images.

Hyperparameter tuning, specifically switching from the Adam optimizer to the SGD optimizer, resulted in significant improvements in the model's performance. However, further exploration of hyperparameter combinations and optimization algorithms could potentially lead to even higher performance.

Additionally, the limitations related to GPU availability in Google Colab and time constraints imposed challenges in training larger models and performing extensive experiments.

References

[1] J. Sivic and A. Zisserman, *Video Google: A Text Retrieval Approach to Object Matching in Videos*, In Proc. ICCV, 2003.

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

[3] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded Up Robust Features," *European Conference on Computer Vision*, pp. 404–417, 2006.

[4] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An Efficient Alternative to SIFT or SURF," *International Conference on Computer Vision*, pp. 2564–2571, 2011.

[5] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, *A Survey on Deep Transfer Learning*, In Artificial Neural Networks and Machine Learning – ICANN 2018, pages 270-279, 2018.

[6] K. He, X. Zhang, S. Ren, and J. Sun, *Deep Residual Learning for Image Recognition*, In Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), pages 770-778, 2016.

[7] Bergstra, J., Bardenet, R., Bengio, Y., & Kégl, B. (2011). Algorithms for hyper-parameter optimization. In *Proceedings of the 24th International Conference on Neural Information Processing Systems* (pp. 2546-2554).

[8] 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.

[9] K. Simonyan and A. Zisserman, *Very Deep Convolutional Networks for Large-Scale Image Recognition*, In Proc. ICLR, 2015.