Mushroom Classification by Genus Using Machine Learning

Laia Querol Alturo

1667878

Abstract—This project presents a machine learning-based approach for classifying mushroom images by genus, a taxonomic rank between family and species. We propose a hybrid model that integrates the Bag of Visual Words (BoVW) [2] framework with color histogram features to improve classification performance. The BoVW method captures local image features by utilizing Scale-Invariant Feature Transform (SIFT) to extract distinctive descriptors, which are then clustered using unsupervised clutering methods such as KMeans to form a visual word vocabulary. To complement these local features, color histograms representing pixel intensity distributions across the RGB channels are incorporated, offering global image characteristics.
For each mushroom image, feature vectors are constructed by combining BoVW representations and color histograms. These vectors are subsequently used to train multi-class classifiers, such as Support Vector Machines (SVM) or K-Nearest Neighbors (KNN). Experimental results demonstrate the effectiveness of this approach, achieving high classification accuracy across 12 mushroom genera. The findings underscore the advantage of combining local feature-based and global color descriptors for robust classification performance.
This system has potential applications in mushroom safety assessments and can serve as a valuable tool for beginner enthusiasts in identifying different mushroom genera.

Keywords—*Bag Of Visual Words (BoVW), SIFT, Computer Vision, Mushroom Genus, SVM, KNN*

Contents

Introduction 1
1 Dataset Exploration 1
2 Descriptor: SIFT vs Dense SIFT 2
2.1 Dealing with NaNs . . . 3
3 Making the Codebook 3
4 Normalization 3
5 PCA 4
6 Considering the Role of Color 4
6.1 Exploring the Effect of Bin Size . . . 4
6.2 PCA with Color Histograms . . . 4
7 Metric Selection 4
8 Tuning Classifier Hyperparameters 4
8.1 k-Nearest Neighbours (k-NN) . . . 4
8.2 Suport Vector Classifier (SVC) . . . 5
8.3 Random Forest Classifier (RF) . . . 5
9 Final Evaluation 5
10 Conclusions 5
11 Acknowledgements 5
References 5

Introduction

Mushroom identification is a popular activity among nature enthusiasts, hikers, and foragers who either intentionally search for mushrooms or encounter them by chance during outdoor excursions. For these individuals, being able to accurately identify mushrooms can be both a rewarding experience and an important safety measure, as some mushrooms are edible while others can be toxic. However, the task of distinguishing between different species and genera can be challenging due to the wide variety of mushrooms and the subtle differences in their appearance. This is where an automated identification system can prove to be quite helpful, providing a reliable and accessible way for individuals to classify mushrooms on the go using their smartphones or other devices.

In biological classification, *genus* is a taxonomic[1] rank that groups together species that share common characteristics. It is one of the key levels in the hierarchical system of taxonomy, falling between family and species. A genus typically includes multiple species that are closely related but differ in certain morphological or genetic traits. For example, the genus `Agaricus` includes various species of mushrooms, such as *Agaricus bisporus* (commonly known as the button mushroom), and `Agaricus arvensis` (the horse mushroom). These well-known species exemplify the diversity within the genus Agaricus. Genus-level classification is an essential step in identifying and studying fungi, as it provides a broader categorization that can be useful for understanding their ecological roles, potential toxicity, and other biological properties.

In this report, we propose an approach that combines both local and global image features to improve the accuracy of mushroom classification. Our method integrates techniques such as the Bag of Visual Words (BoVW) and color histograms to create robust feature representations for mushroom images, which are then used for classification. The following sections will outline the methodology used, present experimental results, and discuss potential real-world applications of this classification system.

This same report, the data and the code can be found at my very own GitHub.

1. Dataset Exploration

To start off, we will have a look at the dataset and show a brief visualization of each of the label classes. Our data consists of 12000 images of size 224x224, evenly distributed across 12 classes. This perfectly balanced dataset is divided into training, validation, and test subsets, ensuring an equal representation for each class. Each class corresponds to the genus of a specific type of mushroom, with the possible genera being: *Cortinarius, Agaricus, Entoloma, Russula, Suillus, Inocybe, Pluteus, Hygrocybe, Amanita, Boletus, Lactarius, Exidia*. See Figure 1. For a non-expert eye, these mushrooms are highly challenging to differentiate, as there is significant variation within each class and notable visual similarity between classes. This inherent complexity not only poses a challenge for human classification, but also highlights the potential difficulty for an algorithm to perform this task effectively.

As you may have already noticed, the mushrooms in the dataset are already segmented. This preprocessing step is highly beneficial as it eliminates potential background noise, allowing the focus to remain solely on the mushrooms. For a closer examination, see Figure 2.

[1] Taxonomy is the branch of biology that classifies all living things. Swedish botanist Carolus Linnaeus developed a classification system called the taxonomic hierarchy, which today has eight ranks from general to specific: domain, kingdom, phylum, class, order, family, genus, and species.[1]

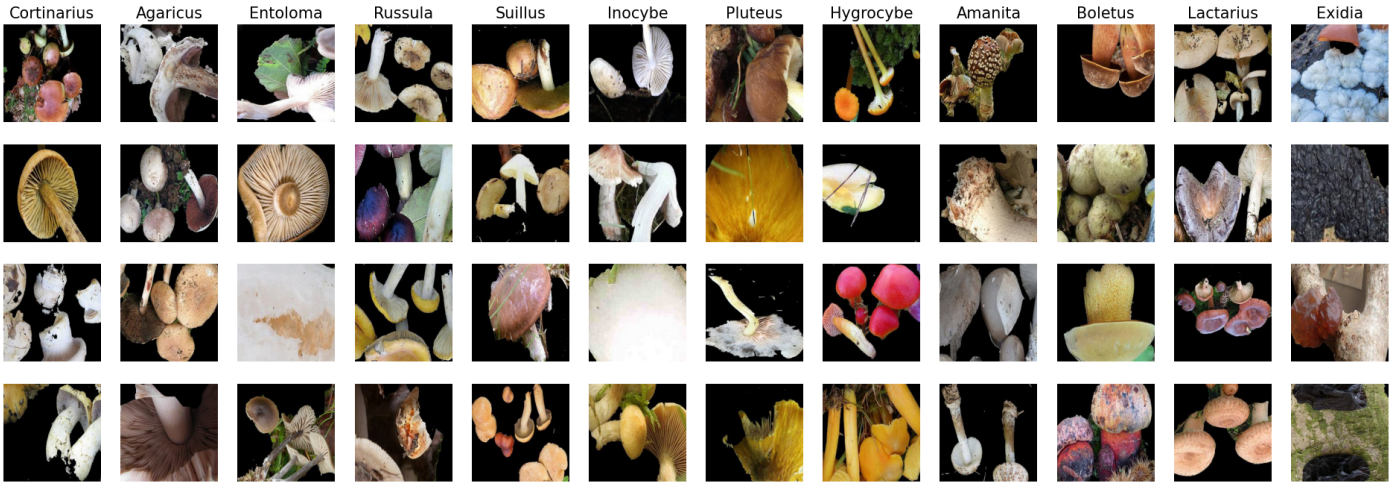


Figure 1. Example visualization of classes.

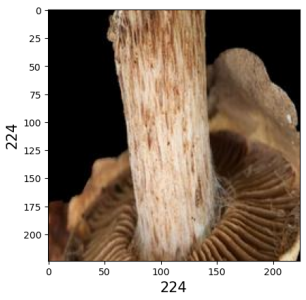


Figure 2. Example mushroom

Since this is a problem of multi-class classification of 12 classes, we will use as a baseline accuracy the one we would be obtaining if we were to use a random classifier, that is $1/12 = 0.083$. Our aim is to improve this accuracy significantly while keeping in mind the scope of the problem at hand.

2. Descriptor: SIFT vs Dense SIFT

The Scale-Invariant Feature Transform (SIFT) detects distinctive keypoints in an image and computes descriptors based on their surrounding regions, making it robust to changes in scale, rotation, and illumination. Dense SIFT, in contrast, extracts descriptors at regular intervals across the entire image, providing a more uniform and exhaustive feature representation. This section explores the differences between these approaches to determine which performs better for our data.

On the one hand, Dense SIFT generates a uniform grid of keypoints across the image. The step size controls keypoint spacing, with smaller steps generating more keypoints for richer detail but at a higher computational cost. Therefore, it is crucial to select a step size that balances sufficient image coverage with efficiency. Figure 3a illustrates the keypoint grids for various step sizes. Notably, many keypoints fall outside the mask, describing areas of no importance. It makes sense to filter these out (Figure 3b) retaining only relevant ones, which not only reduces confusion in our model but also reduces the number of descriptors per image, improving computational efficiency.

On the other hand, standard SIFT identifies keypoints based on their importance, ranking them accordingly. We can adjust the number of keypoints retained by selecting the most significant ones (Figure 3c. While the initial detection and computation remain somewhat constant regardless of the number retained, reducing the descriptors per image lowers computation time during subsequent clustering.

As a baseline for comparing the accuracy and time between the two, we will be using MiniBatchKMeans with `n_clusters = 50` and SVC [5] configured as `decision_function_shape='ovr', kernel='rbf', C=1.0` and `gamma=0.1`. See Table 1. Since the mask can vary across the data, the number of keypoints for Dense SIFT tests has been estimated as 70% of the total points we would have if the mask were not included.

Dense	step	# of kp	Accuracy	Time Elapsed (s)
True	5	1417	0.16	291.55
True	10	370	0.17	204.85
True	15	157	0.18	181.16
True	20	101	0.18	196.58
True	25	56	0.17	177.01
True	30	44	0.16	171.66
True	35	34	0.15	171.59
False	-	16	0.18	205.52
False	-	32	0.19	212.13
False	-	64	0.19	210.86
False	-	128	0.21	215.92
False	-	256	0.18	226.12
False	-	512	0.18	229.40

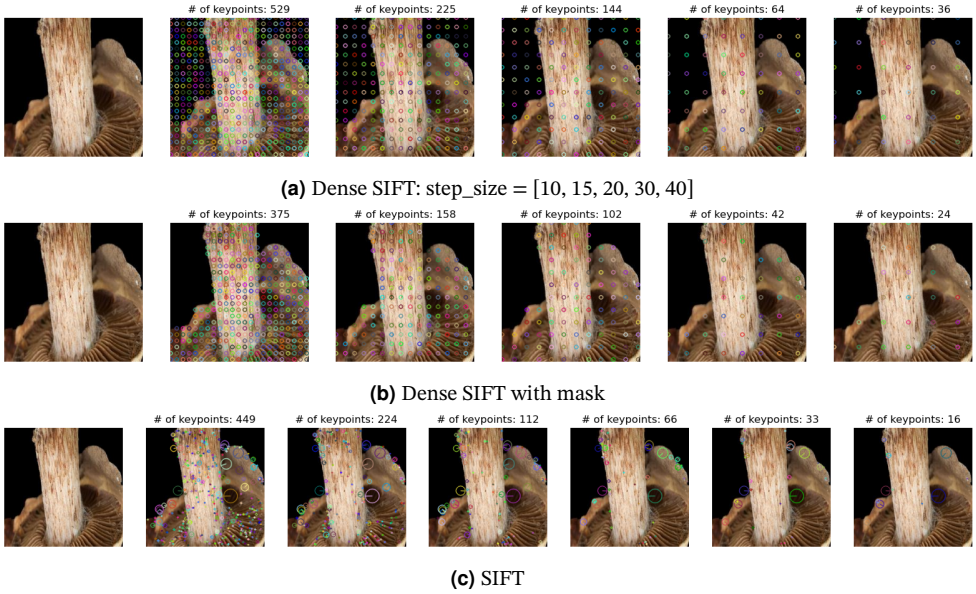
Table 1. Table Comparing Dense SIFT and SIFT

We observe that SIFT achieves the highest accuracy, with a peak value of 0.21 using 128 keypoints. This performance surpasses that of dense SIFT, which maintains an accuracy of approximately 0.18 across different configurations. While the accuracy for SIFT is superior, the processing time is generally longer, which is expected due to the need to detect keypoints. However, MiniBatchKMeans, being a fast clustering algorithm, somewhat masks the fact that increasing the number of descriptors (keypoints) per image—such as in the best dense SIFT configurations—results in longer computational times for clustering. This issue becomes even more pronounced with algorithms like GaussianMixture. With this in mind, let's take a closer look around the optimal parameters found (Figure 2).

Dense	step	# of kp	Accuracy	Time Elapsed (s)
True	14	202	0.18	195.41
True	17	137	0.19	204.76
True	20	101	0.18	192.64
False	-	64	0.19	201.82
False	-	80	0.20	207.36
False	-	96	0.21	208.20
False	-	112	0.20	215.58
False	-	128	0.21	209.51

Table 2. Table Comparing Dense SIFT and SIFT. Final

We conclude that SIFT is the most effective descriptor for our data, as it consistently achieves higher accuracy with a lower number of keypoints. In particular, using 96 keypoints with SIFT yields the



(a) Dense SIFT: step_size = [10, 15, 20, 30, 40]

(b) Dense SIFT with mask

(c) SIFT

Figure 3. Example figure for SIFT and Dense SIFT keypoints visualization.

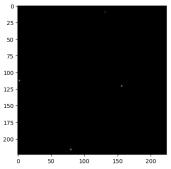


Figure 4. NaN descriptors

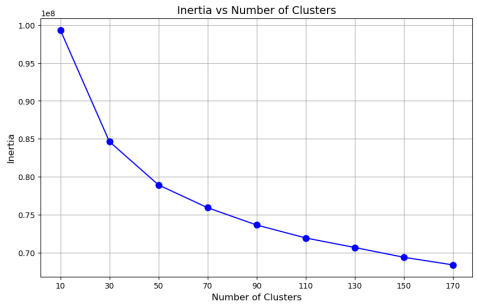


Figure 5. Inertia vs Number of Clusters. MiniBatchKMeans

best performance, striking an optimal balance between accuracy and computational efficiency.

2.1. Dealing with NaNs

In certain scenarios, the SIFT algorithm may fail to detect any keypoints, leading to the absence of corresponding descriptors. With the chosen configuration, we encountered only one such instance, as depicted in Figure 4. This example belongs to the class *Exidia*. However, the image is almost completely black -probably a result of a faulty segmentation- and provides minimal visual information, which results in the failure of the algorithm to extract meaningful keypoints. Since this is an isolated instance, we simply remove it from the dataset. This exclusion has a negligible impact on our pipeline, which now consists of 11,999 (nearly perfectly balanced) data points.

3. Making the Codebook

The next step in our pipeline is to make the *codebook* that will describe our images. This is achieved by clustering the feature descriptors extracted from the training images in the previous section. These descriptors were normalized using StandardScaler in order to guarantee uniformity for clustering.

The codebook serves as a dictionary of visual patterns, where each cluster represents a "visual word." The number of clusters directly determines the size of the codebook, which in turn influences how images are represented overall in our model. Therefore, our goal in this section is to identify the optimal number of visual words to use in the codebook, striking a balance between model expressiveness and computational efficiency.

To achieve this, we employ the MiniBatchKMeans algorithm for clustering. MiniBatchKMeans provides a quick and memory-efficient substitute for conventional K-Means while preserving a similar level of clustering quality, it is especially well-suited for large datasets. We analyze the clustering performance by plotting the inertia[2] values against the number of clusters. To avoid overfitting, it is essential to identify the elbow point in the inertia graph. In Figure 5 we can find it at 50. This will be our codebook size from now on.

With our codebook fitted we can now predict which images contain which visual words and how many. With this information we will make a histogram for each image, which will serve as the features for the classifier. The histograms will be normalized using

4. Normalization

It is always recommended to normalize data so that all features have the same scale. This is known to lead to more robust training and overall better results. We will go through two methods of normalization and see what works best for us.

- Standardization: makes the vectors have 0 mean and unit variance. We use sklearn's `StandardScaler()`.
- L2 norm: makes the vectors have 0 norm. We use sklearn's `Normalizer(norm='l2')`.

We are using `SVC(decision_function_shape='ovr', kernel='rbf', C=1.0, gamma=0.2)` and `KNeighborsClassifier(n_neighbors=7)` as our baseline models.

As shown in Figure 6, L2 normalization yields slightly better performance across both classifiers, although kNN still works great without normalization.

[2] Inertia represents the sum of squared distances between data points and their assigned cluster centroids, serving as an indicator of how tightly the clusters are formed.

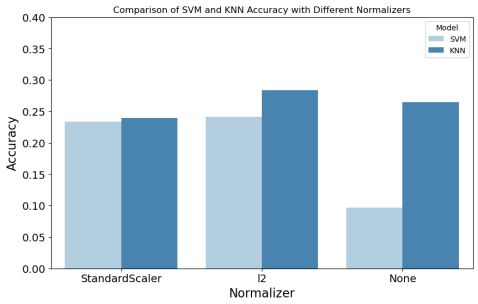


Figure 6. Comparison of Normalization Techniques

5. PCA

Principal Component Analysis (PCA) [3] was applied to the dataset with different values for the number of components, specifically 2, 4, 8, and 16. However, after experimenting with these values, the results indicated no accuracy improvement in the best model from the last section. Additionally, PCA did not highlight any particular features as being more significant than others. We therefore discard it and continue with our 50 visual words.

6. Considering the Role of Color

SIFT is an algorithm designed to be invariant to color, as it computes descriptors using grayscale images. While this works well for many applications where color can vary due to lighting conditions or is not essential for classification, in our case, the color of a mushroom plays a crucial role in its identification. Including color information can significantly enhance the accuracy of classification, especially when distinguishing between different mushroom species.

So far, without incorporating color, the results have not been particularly impressive, suggesting once again that color could be an important factor in improving classification performance. Having our mushrooms already segmented proves of great value in this task as it eliminates the potential noise from the background.

To incorporate color information, we separate the three RGB color channels, each with values ranging from 0 to 255, and then quantize them into a certain number of bins. This results in three color histograms, which we normalize and concatenate with the histogram of visual words to create a richer, more informative feature set for classification.

6.1. Exploring the Effect of Bin Size

Using the same configurations of section 4 for SVC and kNN we obtain the results show in Figure 7. We notice that it for every configuration it almost doubles the accuracy obtained with only the visual words. In kNN we do not see any change in performance between the amount of bins and SVM improves linearly, reaching a top at almost 70% accuracy. We have to keep in mind that the more bins we make, the more features our classifier will have and the more time it will take.

6.2. PCA with Color Histograms

Here we can also try PCA to reduce the number of features. We have seen the best performing number of bins would be 128. Keep in mind that this is applied to each color channel resulting in a total of 384 additional features, which is a lot. It would be beneficial to reduce the number of features while preserving the valuable information that the higher bin count provides.

We applied PCA to histograms with bin sizes of 64, 96, and 128, resulting in 192, 288, and 384 additional features, respectively. For each case, we reduced the dimensionality to 48 components. As shown in Figure 8, the results remain consistent with our previous findings. This demonstrates that PCA has been effective in this context. Specifically, we will retain the bin size of 128, as it offers the best performance while maintaining the same level of dimensionality reduction.

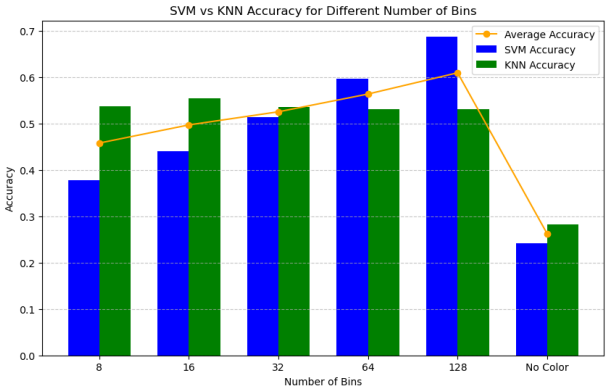


Figure 7. Effect of Bins in Color Histograms

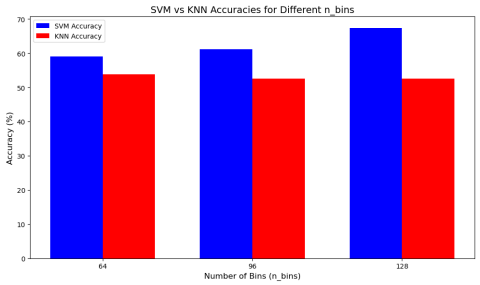


Figure 8. Applying PCA to Color Histograms

To sum up, we have now a total of 98 features that we will use in our classifier.

7. Metric Selection

As we have seen, our dataset is perfectly balanced. This balance ensures that no particular class is overrepresented or underrepresented, making it an ideal candidate for using accuracy as the primary metric for evaluating and comparing the performance of different models, like we have done so far.

8. Tuning Classifier Hyperparameters

Thus far, we have trained our model using classifiers with a configuration that was chosen somewhat arbitrarily. While this approach offeed preliminary insights, it is crucial to assess the model's hyperparameters and their impact on performance. In this section, we will delve deeper into the hyperparameters selected for the SVM and kNN classifiers and explore alternative classifiers that might better suit the task.

For each model, we will use GridSearch with 4-fold cross-validation on the training data. This procedure ensures that we identify the optimal hyperparameters while mitigating the risk of overfitting. Once the best configuration is determined, we will evaluate the performance of each model using the validation set to ensure generalizability.

8.1. k-Nearest Neighbours (k-NN)

The best parameters found were `{'metric': 'manhattan', 'n_neighbors': 3, 'weights': 'distance'}`. Fitting this model with our whole training set and evaluating it with our validation we obtain an accuracy of 78.83%. At Table 3 we can see the classification report. We can observe some differences in f1-score between classes, having the best at class *Boletus* (9) and the worst at class *Amanita* (8).

Class	Precision	Recall	F1-score	Support
0	0.77	0.68	0.72	50
1	0.85	0.78	0.81	50
2	0.66	0.76	0.70	50
3	0.76	0.64	0.70	50
4	0.86	0.84	0.85	50
5	0.79	0.90	0.84	50
6	0.77	0.96	0.86	50
7	0.78	0.86	0.82	50
8	0.74	0.64	0.69	50
9	0.88	0.86	0.87	50
10	0.71	0.74	0.73	50
11	0.93	0.80	0.86	50
Accuracy		0.79		600
Macro avg	0.79	0.79	0.79	600
Weighted avg	0.79	0.79	0.79	600

Table 3. Classification Report KNN

8.2. Suport Vector Classifier (SVC)

The best parameters found were `{'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}`. Fitting this model with our whole training set and evaluating it with our validation we obtain an accuracy of 82.67%. At Table 3 we can see the classification report. We can observe some differences in f1-score between classes, having the best at classes *Inocybe* (5) and *Pluteus* (6) and the worst at class *Cortinarius* (0).

Class	Precision	Recall	F1-score	Support
0	0.65	0.72	0.69	50
1	0.76	0.90	0.83	50
2	0.80	0.82	0.81	50
3	0.77	0.80	0.78	50
4	0.79	0.82	0.80	50
5	0.92	0.90	0.91	50
6	0.92	0.90	0.91	50
7	0.92	0.88	0.90	50
8	0.85	0.80	0.82	50
9	0.88	0.86	0.87	50
10	0.82	0.72	0.77	50
11	0.89	0.80	0.84	50
Accuracy		0.83		600
Macro avg	0.83	0.83	0.83	600
Weighted avg	0.83	0.83	0.83	600

Table 4. Classification Report

8.3. Random Forest Classifier (RF)

A Random Forest (RF) is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control overfitting [4].

The best parameters found were `{'max_depth': 30, 'n_estimators': 200}`. Fitting this model with our whole training set and evaluating it with our validation we obtain an accuracy of 68.00%.

9. Final Evaluation

Now, it is time to evaluate our models using data they have never seen before, the test data. To train these final models we use the best parameters found in the last section with the train and validation set. The test accuracies for the Random Forest (RF), Support Vector Classifier (SVC), and K-Nearest Neighbors (KNN) models are 68%, 83%, and 82%, respectively.

As anticipated, these results closely align with the cross-validation scores. SVC emerges as the top performer, followed closely by KNN, while RF ranks third, also requiring the most computational time. Figure 9 presents the confusion matrices for each model, where the prominent diagonal entries further validate the strong performance of the models.

10. Conclusions

We can conclude that given its speed and accuracy, our best performing model is KNN. An important enhancement to our classification pipeline was the incorporation of color histograms into the Bag of Visual Words (BoVW) algorithm. This addition improved the model's ability to capture color information, which plays a crucial role in differentiating between various mushroom types. By combining both spatial and color features, we achieved better overall classification performance.

11. Acknowledgements

I would like to express my gratitude to my CVC computer, with its 8 powerful cores and 68GB of RAM, for effortlessly handling every task I have thrown at it, without a single complaint. Also, Tau LaTeXtemplate built by Guillermo Jimenez.

References

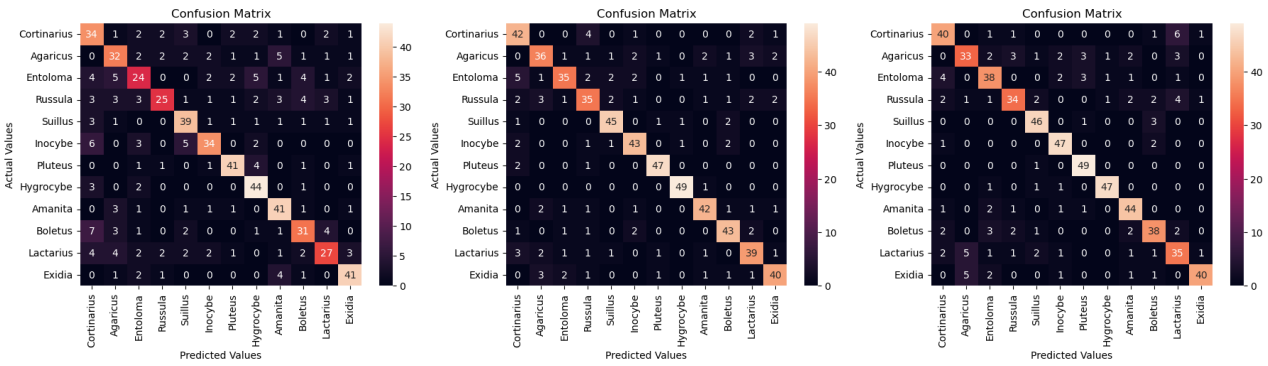
[1] *Biology Dictionary. Taxonomy - Definition*. [Online]. Available: https://biologydictionary.net/taxonomy/.

[2] *Medium. BOVW*. [Online]. Available: https://medium.com/@aybukeyalcinerr/bag-of-visual-words-bovw-db9500331b2f.

[3] *Scikit-Learn. PCA*. [Online]. Available: https://scikit-learn.org/1.5/modules/generated/sklearn.decomposition.PCA.html.

[4] *Scikit-Learn. RF*. [Online]. Available: https://scikit-learn.org/1.5/modules/generated/sklearn.neighbors.KNeighborsClassifier.html.

[5] *Scikit-Learn. SVC*. [Online]. Available: https://scikit-learn.org/1.5/modules/generated/sklearn.svm.SVC.html.



(a) RF **(b)** SVC **(c)** KNN

Figure 9. Example figure for SIFT and Dense SIFT keypoints visualization.