IMAGE CLASSIFICATION WITH SIFT / BoW AND CNN
Exam Project June - July 2023
Supervised Learning
Master Degree in Artificial Intelligence for Science and Technology

Gabriele Canesi
Student ID: 861390

Rebecca Casati
Student ID: 859789

***Abstract*—The goal of this project is to perform image classification on a given dataset, which is a variant of the Tiny ImageNet dataset. The task is solved as a supervised learning problem, with two different approaches.**
As first method it is used a model based on SIFT and Bag of Words (BoW), followed by a Support Vector Classifier (SVC) as a traditional classifier.
The second approach involved using a model based on a Convolutional Neural Network (CNN); to obtain this it was used the REGNET_X_1_6GF pre-trained PyTorch model, of which the final classifier was changed and then trained with transfer learning.
The first model of the two was implemented both in Matlab and Python, while the second one only in Python. For what concerns the first model, although the performances of the Matlab implementation were better (accuracy of 13%), the one that was taken into consideration for the evaluation and the comparison with the CNN based model is the Python implementation.
The SIFT/BoW based model performs with a very low accuracy of $6,9\%$, while on the contrary the CNN performs better, reaching a total accuracy of 70%. Other considerations for evaluating the models were made, leading to the final conclusion that the CNN-based model is clearly better than the other one, and it can still be improved.

1. Introduction

This paper reports the procedure followed and the results found in developing the code for the exam project of the course of Supervised Learning of the Master Degree in Artificial Intelligence for Science and Technology held by the University of Milano Bicocca, the University of Milano and the University of Pavia. The project addressed is the one for the sessions of June - July 2023.
The aim of the project is to perform image classification on a variant of the Tiny ImageNet dataset. The problem was cast as a Supervised Learning problem and two procedures have been followed in order to achieve the classification, with two different Local Descriptors Trends: SIFT (Scale Invariant Feature Transform) [1] with Bag of Words (BoW) [2] coupled with a traditional classifier, and a Convolutional Neural Network (CNN) [3].
The code was implemented both in Matlab and in Python (using Notebooks in Google Colaboratory), in particular the PyTorch framework [4] and the scikit-learn [5] library were used. The Python resources were used for both the classification using SIFT/BoW and the CNN, while Matlab only for the first method of the two.
The paper is organized into sections that deepen the dataset, the two methods used for the classification with the results and their evaluation, and lastly the conclusion.

2. Dataset

The dataset used in this project is a variant of Tiny ImageNet. It contains images 64×64 of 100 different classes. The dataset is already divided into train, validation and test set.
The train set contains 1,000 elements per class, which means a total of 100,000 elements; the validation set contains 100 elements per class, for a total of 10,000 elements; the test set contains 10,000 elements.

The test set was not used, in its place the validation set was used as test set, while the actual validation set was extracted as a subset of the original train set.
For the splitting of the train set into training and validation, the proportions were changed in the different implementations of the model. In all cases a stratified sampling was performed in order to maintain the same class percentage for both training and validation sets; indeed, although the proportions between the different classes were all the same, meaning that probably the same class percentage would have been maintained even with a simple random sampling of the train set, the stratification could help in achieving a better performance. This was done by using already implemented functions both in Matlab [6], [7] and Python [8].

Data Pre-processing

In order to investigate the data, first of all a check was performed in order to know if some gray-scale images were

present, and if the images were coded as RGB, RGBA, BGR or other systems.
It was found that all images are RGB, as desired.

Some transformation were applied to the images after loading the dataset, but a different pre-processing has been applied in the case of the two methods of classification, that will be explained in details in the following sections.

3. SIFT / Bag of Word based method

Bag of words is a technique used in computer vision for object classification and recognition. It is based on three steps: feature extraction, feature clustering (learn the "visual vocabulary" and quantize the local features), and histogram construction for representing the images by frequencies of "visual words".
First of all, key-point detectors and descriptors are used to extract features from the image; then, these features are quantized into a visual vocabulary using a clustering algorithm such as k-means, where the centroids are the "visual words" of the "visual vocabulary". Finally, a histogram is constructed by counting the occurrences of each "visual word" in the image. The resulting histogram serves as representation of the image, and it is used to train a classifier in order to classify the images.

As key-points detector and descriptor SIFT or the improved SURF (Speeded-up Robust Features) [9] can be used.

Matlab

The first choice for implementing the first model of this project (the SIFT/BoW based one) was of using Matlab [10], [11].
The reason behind this choice is the fact that Matlab, unlike Python, provides already implemented functions for the task that allow for easier work and also better results, since these functions have been already studied and improved in order to obtain the best performances.
However, the downside of this is the fact that, exactly because of the existence of these implemented and refined functions, there is close to none flexibility and control by the user of what is actually happening.

In the specific case of this project, the functions used are *bagOfFeatures* [12] and *trainImageCategoryClassifier* [13], which are able to automatically create by themselves the histograms of visual words for each image, and then train a traditional classifier in order to classify them at the end.

Data Loading and Pre-processing. In this case the proportions chosen for the splitting of the train set were 80% of training and 20% of validation, meaning 80,000 images in the train set and 20,000 in the validation set.
No transformation or reshape of the images was performed in this case.

Model. As mentioned before, the function *bagOfFeatures* already performs the features extraction from the images and the creation of histograms of visual words for them.
Looking in detail at what exactly this function does, first of all it extracts the key-points and the descriptors for each image using SURF, then it performs a clustering with k-means of all the descriptors extracted, and, finally, it creates a histogram by inserting each descriptor of each image into a cluster found in the previous step and substituting it with the centroid of that cluster (which is a visual word). The histogram becomes the feature vector for each image.
In this operations only two parameters are free to be changed by the user: the number of clusters of the descriptors for the k-means, and the percentage of strongest (most significant) features to keep after the extraction.

The second function, *trainImageCategoryClassifier*, trains the model starting from the histograms computed before. It is possible to select different types of algorithm to perform the final classification: in this project the model used was a SVC.

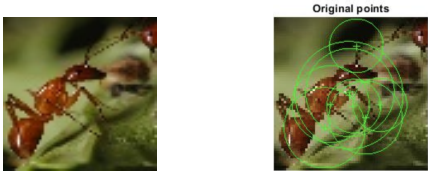
As already mentioned, in the two functions there are some parameters to set, in order to achieve the best performance possible. After some iterations and some trials, these are the parameters that provided the best accuracy (first two are related to the extraction and clustering of the images, while the others to the SVC model):

- number of clusters in which the descriptors are grouped = 500
- fraction of strongest features to use from each label in the clustering = 0.85
- type of kernel function = 'rbf'
- 'BoxConstraint' = 0.1 (smaller means less overfitting)
- 'kernelScale' = 0.6 (higher means more complex model, and more prune to overfitting)

With this configuration, the test accuracy over 10,000 images is 13%.

To report an example of the steps followed in this first part, Figure 1 shows an image from the train set (1a) with its key-points detected with the SURF method (1b); Figure 2 shows the histogram created and associated to the same image.
It is possible to notice in the histogram that one visual word is dominant with respect to the others, that particular word will be the indicator for the classifier that the image belongs to the class associated to that word.

In order to investigate if the low accuracy of the model could be due to overfitting, also the train accuracy was computed.
Indeed, it was found an accuracy of 100% on the train set, meaning a great modelling of the train data but low capability of generalization by the model.



(a) Image (b) Key-points

Figure 1. Example of training image with the key-points found by SURF, with the model implemented in Matlab.

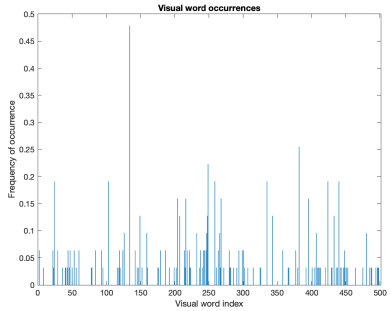


Figure 2. Histogram of visual words associated to the image in Figure 1, with the model implemented in Matlab. On the x-axis there are all the visual words of the visual vocabulary, on the y-axis there is the frequency of occurrence of the visual word.

This can be explained by the fact that all the features extracted from the SURF algorithm are used to search the centroids of the clusters and create the visual words; therefore these features return a detection of very good clusters and create very specific histograms for the training data. Subsequently, the classifier was trained with these very specific histograms and, as a result, it performed a good classification on the train set.
However, the histograms of the test images, with the visual words found with the clustering performed on the descriptors extracted from the train images, are not such good histograms as the ones found for the train images, and this leads to a large decrease of the performances of the model, when evaluated on the test set.
A solution for this problem could be to separate the training set in two parts, one to perform the clustering of the descriptors and the other to train the classifier.

Python

A second and different implementation of the model was done in Python. The reason is the already explained problem that in Matlab there is no possibility of getting hands on the process. Therefore, this part was done mainly for academic purposes, in order to better understand how the classification is performed with SIFT and BoW, coupled with a classifier.

However, in this case it is expected to obtain a worse result with respect to the previous one, due to lack of time resources needed to improve the model in order to obtain the absolute best, which in the previous case was already given in the implemented Matlab functions.

Figure 3. Key-Points detected by SIFT for an image in train set, with the model implemented in Python.

Data Loading and Pre-processing. In this case the train set was divided into 70% of training and 30% of validation, since this proportion was found to give the best results in terms of accuracy of the model.
The data have been loaded with a custom DataLoader [14]. No transformations were applied to the images.

SIFT OpenCV. The features extraction process in this case happens in a different way with respect to what was done with Matlab.
The SIFT algorithm detects the key-points and the descriptors, and then only the second ones are used in the algorithm [15]. An example of key-points extracted by SIFT algorithm is shown in Figure 3.
The descriptors are then saved in two different formats: in the first format they are saved image by image, while in the second format all the descriptors are saved in an unique list of elements, without the distinction of in which image they were found. Both these two formats are important in the next steps in order to initialize the clusters for the k-means algorithm and to create the histograms for each image.

Mini-Batch K-means clustering of the features. All the descriptors that have been extracted in the previous step, saved as a list in the second format described above, are used to initialize the clusters (visual words) in order to, subsequently, create the histograms.
The aim of the clustering is to regroup all the features that describe the same aspect of an image, in order to create a single ”word” to describe better the image and to reduce the large number of features, otherwise difficult to handle by the classifier in the next step. The best accuracy is achieved with 2,000 clusters.
To perform this step, Mini-batch k-means [16] is used as clustering algorithm because it guarantees a faster execution with the same precision as traditional k-means.

After finding the visual words, the descriptors of each image (saved in the first format previously described) are associated to a cluster and then substituted with the centroid of the cluster to which they belong (which is the visual word of the vocabulary). In this way histograms are generated, containing information about the kind of features present in each image; in particular, each histogram shows the frequency of each visual word, and these visual words are

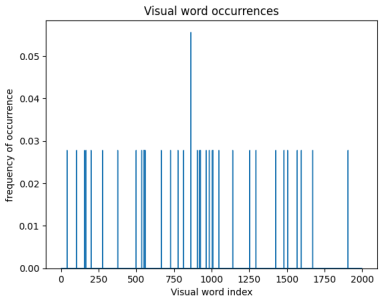


Figure 4. Histogram created for an image in the train set, with the model implemented in Python.

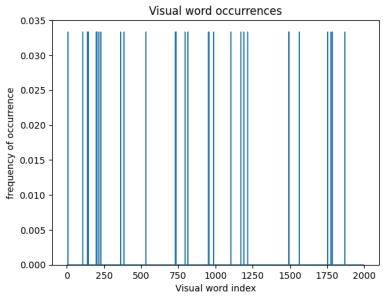


Figure 5. Histogram created for an image in the test set, with the model implemented in Python.

just the 2,000 centroids. This representation is more compact and can be seen as a feature vector of the image. Therefore, each image is converted into the associated histogram before passing it to the classifier.
In Figure 4 is reported an example of histogram obtained in this step (notice that this histogram is not the one associated to the image shown in Figure 3, because during the training a shuffle is performed, which changes the indices of the images, making almost impossible to pick again the same image). Also this histogram, like the one shown in Figure 2 found in Matlab, shows one dominant word with respect to the others, which will lead the image to be classified into the corresponding class.

Traditional Classifier. The last step of this procedure is to train a traditional classifier (SVC, Random Forest, etc.) that takes as inputs the histograms created before and the true labels for each image, and estimates the best model to achieve the classification task.
Different types of classifiers were tested, and the best results are obtained with a SVC using a rbf (radial basis function) kernel [17].

Testing. The testing procedure follows the same steps as the training one: features are extracted with SIFT algorithm from the 10,000 test images, then, using the same clusters initialized before in the training, each descriptor is associated to its nearest neighbor in the dictionary of ”visual words” (centroids of the clusters of features); finally, histograms are created for each test image and classified by the trained classifier to predict the class of each test image.

Figure 5 shows an example of histogram associated to a test image. This histogram is more noisy with respect to the example reported for a train image (Figure 4); indeed, it shows that there are a lot of visual words with the same frequency, and this could make it difficult for the classifier to correctly predict the class of the image.

Evaluation. In order to better investigate the performances of the model, and understand which are its strengths and weaknesses, some metrics and graphs were computed.

First of all, the total accuracy was computed, finding a value of 6.9%. As expected, this value is much lower than the Matlab one (it is about the half); indeed, as already explained, Matlab's algorithm presents a better optimization for this kind of tasks.

To analyze more in depth the model, also the accuracy for each class was computed: it is displayed in the graph shown in Figure 6. It is possible to notice from the graph that there are some classes with an accuracy of 0% (e.g., 37, 42, 51, 66, 70), meaning that the model never recognises images belonging to these classes, while the best accuracy is found for class 23 with a value of 17%.
A possible explanation of why the class 23 has the highest accuracy, in particular with respect to the others, can be due to the labels of these classes: class 23 corresponds to ducks (example shown in Figure 7a), while for example class 37 corresponds to Chihuahuas (example shown in Figure 7b). Indeed, it is plausible to suppose that the features extracted in the case of class 23 are quite general and found in many other classes, therefore this model has troubles separating classes that are similar.
But also, among the classes not recognised at all (0% of accuracy), many are classes of insects, like class 51 is the class of flies (example shown in Figure 7c). In this case the reason behind the failure of the model could be the fact that the resolution of the images in input is too low for extracting appropriate features for such small objects. As a result, the classifier struggles to understand to which class these features belong and consequently it misses completely some classes.

As last evaluation, the confusion matrix [18], shown in Figure 8, can help to get a complete vision of the model.
In particular, besides the expected dark diagonal line, it is possible to see a darker vertical line, in correspondence of classes 23-25, meaning that the model predicts this class very often, even when the ground truth is completely different from these classes; that is why class 23 has the highest accuracy, as already said before. But also some other dark vertical lines are visible, meaning that also for other classes (the ones with higher accuracy, that can be seen from the bar graph in Figure 6) happens that they are

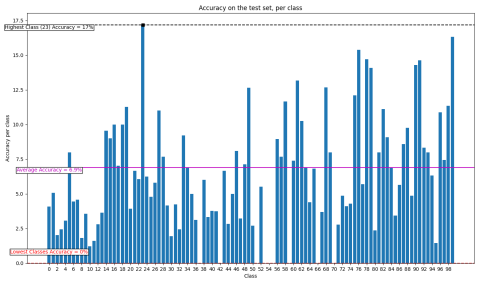


Figure 6. Bar chart of the accuracy of the SIFT/BoW based model implemented in Python on the test set, per class. The red dashed line corresponds to the lowest class accuracy, the black one corresponds to the highest one, and the magenta solid line corresponds to the average value of accuracy for all the classes.

(a) Class 23 (b) Class 37 (c) Class 51

Figure 7. Examples of images of the classes 23 (duck), 37 (Chihuahua), and 51 (fly) of the test set.

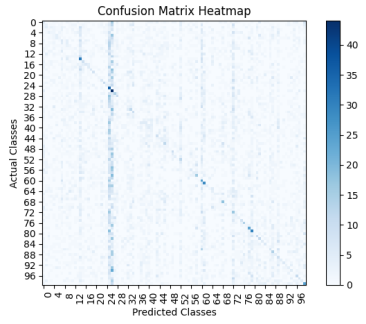


Figure 8. Heatmap for the SIFT/BoW based model implemented in Python.

predicted often, even when the image does not belong to that class. This is in agreement to the hypothesis that this model extracts general features from the images.

After all this evaluations, it is possible to conclude that the model implemented is biased in favor of class 23, and for this reason it shows an higher value of accuracy.
A way to improve the model could be to find a way to extract more specific features from the train images, in order to make clearer the difference between the various classes.

Like in the Matlab implementation, actually, the main problem of this model is overfitting. This hypothesis was confirmed after computing the accuracy of the model on the train set: it was obtained 70% of accuracy, which is a lot higher with respect to the value of 6,9% found on the train set.
Indeed, as it is also clear from Figure 5, is that the histograms generated by the model for the test images are too noisy and are not as clearly categorizable as the ones generated for the train images (exactly as it also happens in Matlab), and this is what can throw off the SVC, which consequently is not able to correctly classify the test image, leading to the very poor performance.

4. CNN based method

Convolutional Neural Networks are a sub-type of Neural Networks, their architecture is a deep feed-forward one.
CNNs are the most used architectures in computer vision and in particular for image classification; indeed, they have achieved state-of-the-art performance on many image classification benchmarks.

CNNs are suited for image classification because with deep learning they are able to learn the spatial hierarchies of features, starting with simple patterns such as edges and moving on to more complex patterns (more global and invariant features) as the layers of the net get deeper.
This hierarchical feature learning is very well suited to image classification, because the visual features of an image can vary a lot.
The patterns and features in the images are recognised by CNNs through their convolutional layers, which apply a set of filters (filter bank) to the inputs that detect specific patterns (each filter creates a new channel in depth in the output of the layer). [19]

Pre-processing and Data Loading. The splitting of the train set was performed keeping 80% of train and 20% of validation.
The data have been loaded with a custom DataLoader [14] in batches of 16 images for the training and validation set, and one by one for the test set.
Some transformation have been applied to the images when loaded, to all three train, validation, and test set:

- Resizing to 232×232
- Central crop to 224×224
- Conversion of the image from ndarray to tensor in the range $[0.0, 1.0]$
- Normalization of the pixel values with $mean = [0.485, 0.456, 0.406]$ and $std = [0.229, 0.224, 0.225]$.

These transformations have been applied in order to adjust the images for feeding them to the CNN.

Model. As CNN it was chosen to use a pre-trained TorchVision model. This is why the transformations listed above were applied, because the chosen model was trained on images with those specifications. Obviously the transformations depend on the pre-trained model that is

TABLE 1. PROPERTIES OF REGNET_X_1_6GF_WEIGHTS.IMAGENET1K_V2

Acc@1	79.668
Acc@5	94.922
Params	9.2M
GFLOPS	1.6

```
(fc): Sequential(
  (0): Linear(in_features=912, out_features=500, bias=True)
  (1): ReLU()
  (2): Linear(in_features=500, out_features=250, bias=True)
  (3): ReLU()
  (4): Linear(in_features=250, out_features=100, bias=True)
)
```

Figure 9. Fully connected classifier of the net.

used.

The model used is REGNET_X_1_6GF, which has the RegNet architecture [20], with IMAGENET1K_V2 as weights [21]. The properties of this model are shown in Table 1 (accuracies are reported on ImageNet-1K using single crops).

The classifier of the model was changed manually with three fully connected layers with ReLU (Rectified Linear Unit) as activation function, and imposing 100 output neurons (one neuron for each class that needs to be detected). The structure of the classifier is shown in Figure 9.

Training (Transfer Learning). Exploiting transfer learning, the model was trained by freezing the parameters of the CNN in order to train only the classifier.
It was chosen to perform this type of training because the dataset on which the classification has to be performed is a smaller variant of the ImageNet dataset, on which the PyTorch model was originally trained, therefore the two datasets are very similar.

The training was performed for 10 epochs, imposing a patience of 8 and saving the model obtained at the last epoch.
The trend of the Validation Loss with the number of epochs is shown in Figure 10. The image shows the expected behaviour characterized by a rapid decrease and the reaching of a plateau for an high number of epochs, however it is also noticeable an oscillating trend.
This oscillating trend could be due to the difficulty of the model of correctly separating the different classes.

It was used the Cross Entropy as loss function, Adam as optimizer with learning rate $lr = 0.001$, adjusted by the cosine annealing scheduler with $T\_max = 100,000$ and $eta\_min = 10^{-5}$.

This particular model was chosen among some others that were tried but obtained a worse, albeit similar, accuracy in the evaluation. The models tested are:

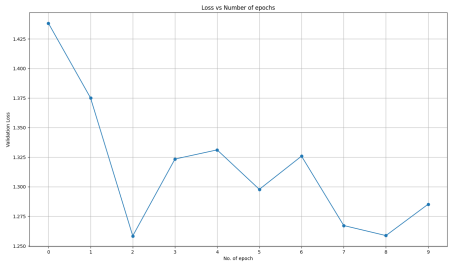


Figure 10. Validation Loss trend with the number of epochs.

- ResNet18_Weights.IMAGENET1K_V1
- MobileNet_V3_Large_Weights.IMAGENET1K_V2
- EfficientNet_B0_Weights.IMAGENET1K_V1

These models and the one used have been found in the PyTorch models documentation [22]. They were chosen because of their relatively low number of parameters, which allowed for a non-prohibitive training time, combined with their high accuracy, in order to obtain good results.

Some hyper-parameter tuning was also performed in order to obtain the best performance achievable. This was done by fine-tuning by hand the values of the following parameters :

- Proportion of the training and validation set
- batch size (for both training and validation set)
- weights of the pre-trained model
- fully connected classifier's layers, neurons and activation functions
- optimizer
- learning rate and parameters of the scheduler (T_max and eta_min)

The values that gave the best results have been already reported above.

Evaluation. In order to evaluate the model, the accuracy of the classification on the test set was computed: the total accuracy is 70%.
It was also computed the accuracy for each class, and it was found that this value varies greatly depending on the class; indeed, the lowest accuracy is found for class 55, equal to 33%, while the highest value is 95%, found for class 95.
Figure 11 shows the values of class accuracy with a bar graph, the values of lowest, highest and average accuracy are highlighted with colored lines.

From these results it can be deduced that the performance of the model depends largely on the class that it is trying to detect, and that the difference in accuracy values in different classes is remarkable. It is possible to understand why it is so by looking at the classes that correspond to the lowest and highest values of accuracy.
Class 55 is the class of stick insects (an example is shown

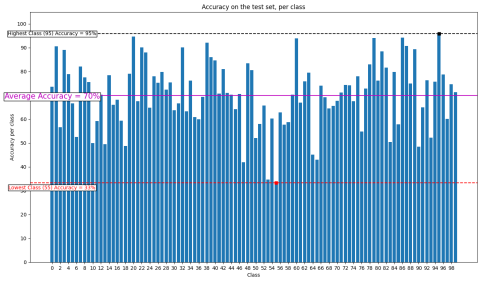


Figure 11. Bar chart of the accuracy of the CNN based model on the test set, per class. The red dashed line corresponds to the lowest class accuracy, the black one corresponds to the highest one, and the magenta solid line corresponds to the average value of accuracy for all the classes.

(a) Class 55 (b) Class 95

Figure 12. Examples of images of the classes 55 (stick insect) and 95 (canoe) of the test set.

in Figure 12a), that are known to be very small and slim, which makes it difficult to detect and correctly classify them; while class 95 is the class of canoes (an example is shown in Figure 12b), which are a category of boat clearly distinguishable, therefore easy to classify, which leads to obtain 95% of accuracy (classification performed correctly in almost all the cases).

Another evaluation that was made in order to have a deeper understanding of the model performances is the generation of the heatmap confusion matrix [18], shown in Figure 13.
This matrix shows good results since it reports darker colours on the diagonal, and lighter ones outside: this means that most of the times the model predicts correctly the class. Since the dataset contains a lot of classes, the differences in the individual class accuracies are not clearly visible, but with a careful look it is possible to notice that the squares on the diagonal corresponding to the classes with higher accuracy are dark-blue, while the ones corresponding to the classes with the lowest values of accuracy are light-blue.

The average value of accuracy of 70% is not bad but it is not great either, although after many attempts it was still the best result achieved. However, the authors do not exclude the possibility of finding a learned model with better performance, for example by increasing the number of epochs.
Another attempt that could be done to improve the model is to fine-tune it, in order to update all the weights of the CNN, or even to extract the features at a deeper level (lower

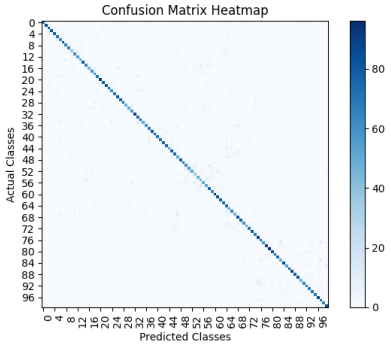


Figure 13. Heatmap for the CNN-based model.

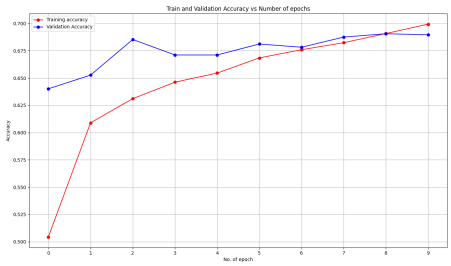


Figure 14. Train and Validation Accuracy with the number of epochs.

layers) with respect to what was done in this paper, in order to have access to more general features, less specific for the original dataset on which the model was trained on.
In order to find another way to improve the model, it was considered if it would have made sense to do data augmentation to avoid overfitting, since overfitting is a common problem for CNNs because they have many parameters. To find if the model overfitted the data, the training and validation accuracy values were studied, as a function of the number of epochs in the training; what was found is shown in Figure 14. From the plot it can be inferred that there is no overfitting, therefore this is not a problem of this model.

5. Last Evaluation: prediction on images generated by a diffusion model

As last evaluation of the two models, in order to analyze how they perform and which types of features are more specific with respect the others, new images were given to the two models to be classified. These images have the peculiarity of belonging to multiple classes at the same time; they are generated with the help of a common diffusion model, like the ones that can be found on the internet [23]. It was chosen, at random, to generate eleven images containing both a cat and a dog; therefore, they would belong to classes from 37 to 42 (different breeds of dogs and felines). Figure 15 represents three examples of these images.
The two models gave different results:

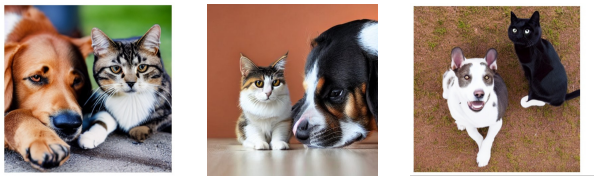


Figure 15. Examples of images generated with the diffusion model [23].

- The SIFT/BoW based model's predictions [79 37 25 25 52 25 25 25 89 72 75] are very imprecise. Indeed, the model recognises the class of a dog only once (class 37); all the other times, instead, predictions are wrong. In particular, the most predicted class is 25 (koala) and this observation reflects the result highlighted by the confusion matrix 8, where there was a dark vertical line around the classes 23-25. With this trial, it is confirmed that the model is biased in favor of those classes. This allows to confirm the already stated conclusion that the features extracted by this model are quite general, and not specific for each class, leading to the poor performance.

- The CNN based model's predictions [38 39 40 91 37 38 91 37 38 40 37] are more accurate with respect to the ones of the previous model. In general, this model recognises and classifies the images always as a dog (as classes of different breeds of dogs), with a level of confidence pretty high (most of the times greater than 50% of probability), but twice the class predicted is 91, which is completely wrong since it is the class of bow ties (fun fact: in Italian the bow tie is also called "papillon", which, absurdly, is also a breed of dogs; but this is obviously just an unfortunate coincidence). In general, these results are consistent with what was already written in the evaluation part of this model (section 4); indeed, classes 37 and 38 have the highest accuracy among the other classes considered in this test (this can be seen from the bar graph in Figure 11), and this means that from the images of these classes the model is able to extract more specific features, meaning that it has more probability of recognising the dogs (classes 37 to 41) with respect to felines (classes 42 to 44).

6. Conclusion

The aim of this project was image classification on a smaller variant of the Tiny ImageNet dataset. Two different models were implemented for this purpose: a SIFT/BoW-based one, implemented both in Matlab and Python, and a CNN-based one, implemented only in Python.

For what concerns the SIFT/BoW-based model, there is a big difference between the implementation in Matlab and the one in Python in terms of performances: the accuracy of the model in Python (6.9%) is about the half of the one implemented in Matlab (13%).
Despite the significant difference in the values of accuracy, both of the models do not perform well, since both values are very low.
The difference in the values of accuracy is due to the fact that in Matlab there were available some implemented functions, already improved for the task, with very low parameters that the user could modify in order to further improve the model. This, however, suggests that the value of accuracy of 13% is probably the best that can be achieved for this task on the given dataset.
On the contrary, the implementation of the model in Python, even though less performing than the Matlab one, allowed to build by hand and work directly on the steps that the model follows to perform the classification, resulting in a deeper understanding of the bag-of-words model.
The huge problem of this model in both implementations is overfitting.

On the other side, the CNN-based model showed an accuracy of 70%, which is much greater than the previous ones, even though it could definitively still be improved.

The results just reported lead to think that, for future researches and improvements for image classification in this direction, probably the SIFT/BoW-based method should be left behind, and more focus should be put into improving the CNN-based method, that has good premises for performing a great classification. This, for example, could be achieved with some more hyper-parameter tuning, made by hand or with a more systematic method.
Unfortunately, this has not been done in this project because of the amount of work that was already required to be done in a time that necessarily had to be limited.

Acknowledgments

The authors would like to thank professor Simone Bianco and Mirko Agarla, that held the course of Supervised Learning, for their clear explanations and willingness to offer clarifications when requested.

The authors are fully aware of what plagiarism is and of its consequences, therefore they hereby declare that the contents of this paper are original. Both the code and report parts were written only by the authors, and all references and sources used were properly cited.

References

[1] Lowe, D.G. "Distinctive Image Features from Scale-Invariant Keypoints". International Journal of Computer Vision 60, 91–110 (2004).

[2] L. Fei-Fei and P. Perona, "A Bayesian hierarchical model for learning natural scene categories". 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), San Diego, CA, USA, 2005, pp. 524-531 vol. 2.

[3] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner. "Gradient-based learning applied to document recognition". in Proceedings of the IEEE, vol. 86, no. 11, 2278-2324 (1998).

[4] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... and Antiga, L. (2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library." In Advances in Neural Information Processing Systems (pp. 8024-8035). pytorch.org: https://pytorch.org/.

[5] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... and Vanderplas, J. (2011). "Scikit-learn: Machine Learning in Python." Journal of Machine Learning Research, 12, 2825-2830. scikit-learn.org: https://scikit-learn.org/.

[6] MathWorks. "ImageDatastore." MATLAB Documentation: https://it.mathworks.com/help/matlab/ref/matlab.io.datastore.imagedatastore.html.

[7] MathWorks (2023). "ImageDatastore.splitEachLabel." MATLAB Documentation: https://it.mathworks.com/help/matlab/ref/matlab.io.datastore.imagedatastore.spliteachlabel.html.

[8] scikit-learn (2023). "sklearn.model_selection.train_test_split." Scikit-learn Documentation: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html.

[9] Bay, H., Tuytelaars, T., Van Gool, L. (2006). "SURF: Speeded Up Robust Features". In: Leonardis, A., Bischof, H., Pinz, A. (eds) Computer Vision – ECCV 2006. ECCV 2006. Lecture Notes in Computer Science, vol 3951. Springer, Berlin, Heidelberg.

[10] Hesham Eraqi (2023). Image Classification with Bag of Visual Words (https://www.mathworks.com/matlabcentral/fileexchange/62217-image-classification-with-bag-of-visual-words), MATLAB Central File Exchange. Retrieved May 11, 2023.

[11] Image Classification with Bag of Visual Words. Csurka, G., C. R. Dance, L. Fan, J. Willamowski, and C. Bray. Visual Categorization with Bags of Keypoints. Workshop on Statistical Learning in Computer Vision. ECCV 1 (1–22), 1–2. https://it.mathworks.com/help/vision/ug/image-classification-with-bag-of-visual-words.html.

[12] MathWorks (2023). "bagOfFeatures." MATLAB Documentation: https://it.mathworks.com/help/vision/ref/bagoffeatures.html.

[13] MathWorks (2023). "trainImageCategoryClassifier." MATLAB Documentation: https://it.mathworks.com/help/vision/ref/trainimagecategoryclassifier.html.

[14] PyTorch (2023). "Data Loading and Processing Tutorial." Tutorial PyTorch: https://pytorch.org/tutorials/beginner/data_loading_tutorial.html#dataset-class.

[15] Introducton to SIFT - OpennCV https://docs.opencv.org/4.x/da/df5/tutorial_py_sift_intro.html.

[16] scikit-learn (2023). "sklearn.cluster.MiniBatchKMeans." Scikit-learn Documentation: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.MiniBatchKMeans.html.

[17] scikit-learn (2023). "sklearn.svm.SVC." Scikit-learn Documentation: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html.

[18] scikit-learn (2023). "sklearn.metrics.confusion_matrix." Scikit-learn Documentation: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html.

[19] datagen.tech (2023). "Image Classification using CNN." DataGen: https://datagen.tech/guides/image-classification/image-classification-using-cnn/.

[20] Radosavovic, I., Dollár, P., Girshick, R., He, K., and Girshick, B. (2020). "Designing Network Design Spaces". arXiv preprint arXiv:2003.13678.

[21] PyTorch (2023). torchvision.models.RegNet_X_1_6GF_Weights. PyTorch.org: https://pytorch.org/vision/main/models/generated/torchvision.models.regnet_x_1_6gf.html#torchvision.models.RegNet_X_1_6GF_Weights.

[22] PyTorch (2023). torchvision.models. PyTorch.org: https://pytorch.org/vision/main/models.html.

[23] Stable Diffusion Online (2022): https://stablediffusionweb.com/#demo