M3 Project: Hand-crafted vs Learnt Features in Image Classification

Ignacio Galve Ceamanos, Brian Guang Jun Du, and Eric Henriksson Martí

Autonomous University of Barcelona, 08193 Bellaterra, Barcelona
{ignacio.galve, brian.guang, eric.henriksson}@autonoma.cat

Abstract. Several models are devised to address the problem of image classification and compared with each other. The use of local descriptors as hand-crafted features through the implementation of a Bag of Visual Words approach is initially explored. Both k-nearest neighbors and support machine vectors are tested as classifiers within this system, and the benefits of dimensionality reduction and spatial pyramid techniques are estimated. Next, a series of learning-based models are constructed and evaluated as additional alternatives. Simple multilayer perceptrons are first addressed, and the possibility of improving their performance through patch-based methods and through their combination with other classifiers is considered. Convolutional neural networks are tested afterward, both through the use of the existing *MobileNetV2* model via transfer learning and through the construction of a new model from scratch. All models are subject to different kinds of parameter fine-tuning to better understand their behavior, increase their classification performance, and maximize their compactness.

Keywords: Image classification · Machine learning · Bag of visual words · Multilayer perceptrons · Convolutional neural networks

1 Introduction

The purpose of the M3 project reflected in this report is that of testing and understanding different kinds of models capable of classifying images into a given set of categories. Image classification is in fact one of the most common subjects in computer vision, as it can be beneficial in a wide variety of applications ranging from medical diagnostic to the automated censorship of inappropriate content.

In this specific case, the idea is to correctly classify images of landscapes into one of 8 different categories: "Open country", "Coast", "Forest", "Highway", "Inside city", "Mountain", "Street" and "Tall building". Some examples of pictures for each of these categories are displayed in Figure 1. The models need to be capable of establishing relationships between certain features in training images and the labels assigned to these in a way that allows them to later carry out predictions on testing images.

A dataset comprised of 1881 training images and 807 testing images is available for the proposed project. Table 1 details how many images belong to each category within the training and test sets.

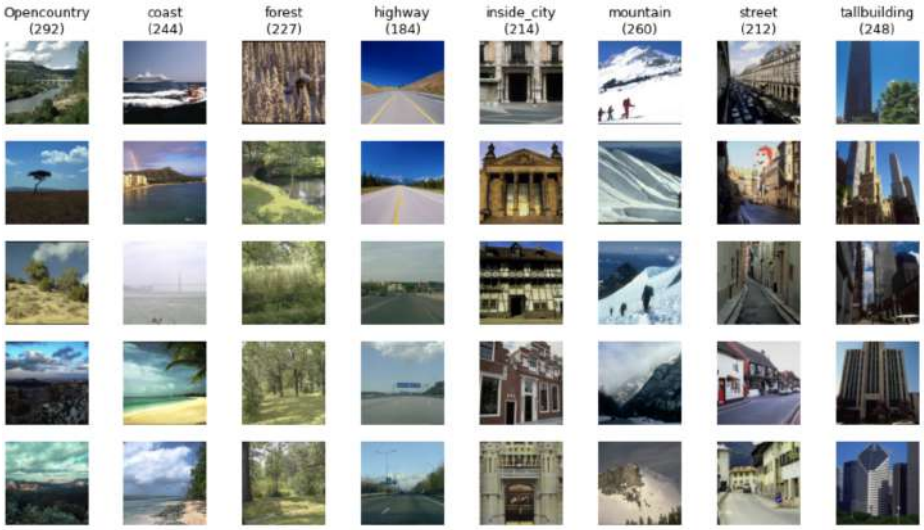

Fig. 1: Samples of images for each considered category.

Table 1: Overview of the dataset.

Category	**Training images**	**Testing images**
Open country	292	118
Coast	224	116
Forest	227	101
Highway	184	76
Inside city	214	94
Mountain	260	114
Street	212	80
Tall building	248	108

Models based on hand-crafted features are first studied and tested on the presented dataset. These are built on the bag of visual words (BoVW) approach[1], where features are basically "words" that represent groups of local descriptors with similar characteristics. Next, fully-connected multilayer perceptrons (MLP) are explored, followed by the more effective convolutional neural networks (CNN). Unlike the BoVW-based models, these two kinds of models are meant to learn which features to consider on their own.

2 Bag of Visual Words (BoVW)

The general concept of BoVW models[1] built for this project is that of extracting local descriptors from images, creating a codebook of visual words by grouping these into clusters, and representing images as histograms reflecting the occurrence of these visual words. A classifier is then trained to interpret these histograms and relate them to one of the 8 categories considered in the classification problem at hand. The next subsections detail the steps followed to implement these models, as well as the results obtained from these.

2.1 Computing key-points and local descriptors

Two main approaches are considered when it comes to extracting local descriptors. First, algorithms like SIFT[2] and AKAZE are used to detect key-points in images and create descriptors. Examples of these can be observed in Figure 2. While using these algorithms is rather simple due to how they do not require tuning any parameters, the amount and the distribution of local descriptors resulting from using them can vary a lot from case to case. This can derive in some images with large smooth areas being represented only through a few relevant details present in them, like it is the case for the images displayed in Figure 2.

Dense SIFT[2][3] is proposed as an alternative to deal with the mentioned issue. It consists in placing key-points in fixed locations over the entire image, usually based on a grid-like pattern defined by a step value. This ensures that all images, no matter their characteristics, are represented by an equal amount of uniformly-distributed descriptors. The local descriptors built on the established key-points can be set to one specific scale or to several ones. An example of dense SIFT computed on the same image as in the previous figure is shown in Figure 3.

Even though in dense SIFT key-points are not detected the same way as with regular SIFT, the way of computing the local descriptors is exactly the same. Local descriptors are therefore also represented as orientation invariant 128-dimensional arrays.

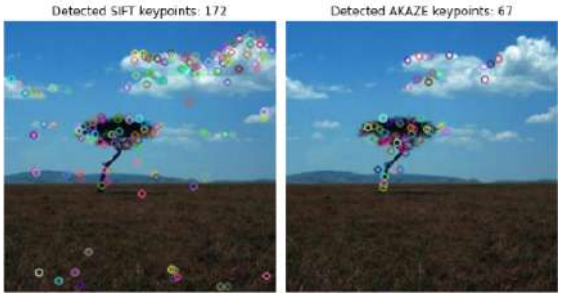

Fig. 2: Key-points using SIFT and AKAZE algorithms.

Fig. 3: Key-points using dense SIFT with a step of 5px and a scale of 3px.

2.2 Constructing a codebook

Once the descriptors have been computed, it is time to construct a codebook[1] that can simplify features by assigning them to certain clusters or visual words. *k-means* clustering is used to create this codebook from the local descriptors of the training images in an unsupervised way. The parameter k in said method determines the size of the codebook being constructed.

Having the codebook ready, images can now be represented through an histogram that details the amounts of each kind of visual words they contain.

2.3 k-NN classifier

The k-nearest neighbors algorithm (k-NN) has first been considered as the mean of interpreting the computed histograms and assign images to their closest category. This algorithm simply carries out classification by checking the category of the k nearest neighbors of the addressed image and assigning the most common one to it.

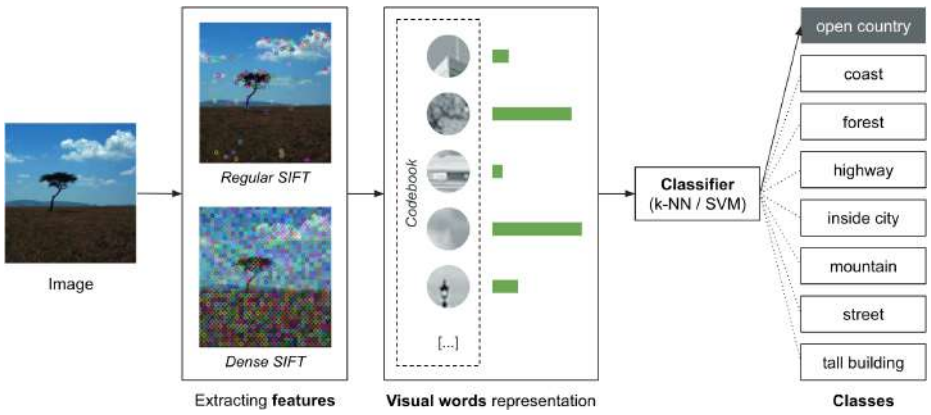

Fig. 4: General overview of the BoVW approach.

Several tests have been carried out using this classifier under different conditions in an effort to determine the influence of aspects such as the size of the codebook, the k value of the k-NN, or the distance metric on the performance of the model. It is worth noting that results marked as using dense SIFT have used a local descriptor size of 3px and a step value of 5px.

Testing different codebook sizes As one can observe in Figure 5, for the model built on the normal SIFT, the accuracy peaks at a codebook size of 128. From that point onward, it starts decreasing in almost a linear fashion. On the other hand, the model built on the dense SIFT showcases a different behavior, as its accuracy remains almost even for all of the tested codebook sizes. While it is hard to pinpoint exactly why this occurs, it is hypothesized that these different behaviors have to do with how the descriptors computed using dense SIFT are more diverse and capable of describing all kinds of image to a decent extent, while the ones from the normal SIFT focus on detail and are of little use on simpler images.

Testing different *k* values for the k-NN Based on what can be observed in Figure 6, other than for excessively low amounts (below 5), increasing the amount of neighbors for the k-NN classifier does not seem to lead to any major improvements in accuracy. Using around 10 neighbors appears to be the sweet spot. That being said, the general takeaway is that very low numbers of neighbors lead to rather unstable decision boundaries, while after a certain threshold the performance of the classifier reaches a plateau. If higher amounts of neighbors had been considered, it is very likely that the accuracy would have eventually decreased due to most points falling into a dominant class.

Testing different distance metrics for the k-NN As seen in Figure 7, the Manhattan and Braycurtis distance metrics seem to lead to the best performance if both normal and dense SIFT approaches are taken into account.

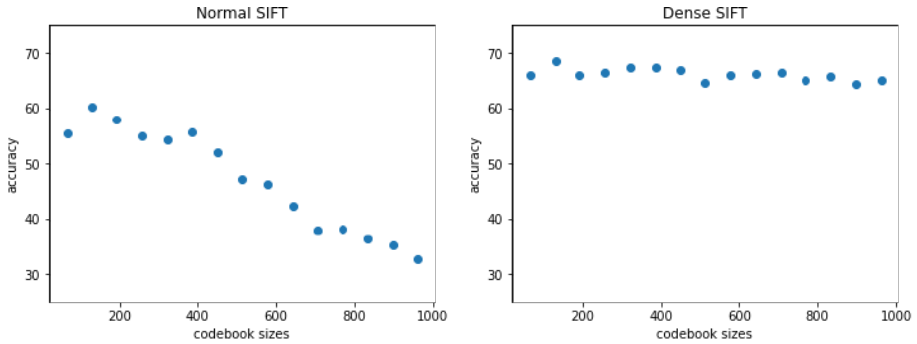

Fig. 5: Effect of codebook size on accuracy for normal and dense SIFT.

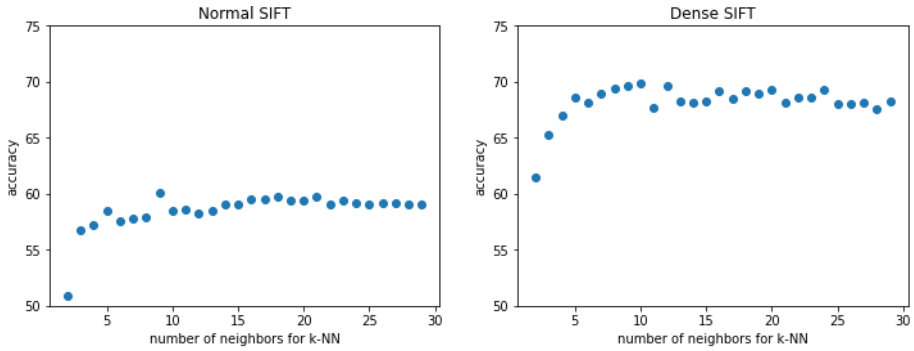

Fig. 6: Effect of k neighbors on accuracy for normal and dense SIFT.

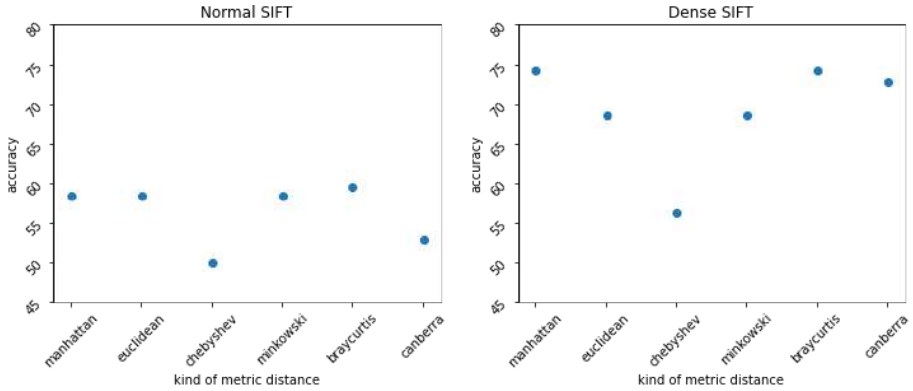

Fig. 7: Effect of various distance metrics on accuracy for normal and dense SIFT.

If all of the tested parameters are set to their optimal values, the accuracy of the k-NN based classifier reaches a value of 75.59%. In order to evaluate how this performance would extend beyond the considered set of data, 10-fold cross validation is considered to recalculate the score. Implementing this cross validation brings the model's accuracy to an average of 74.85%, which indicates that it should technically maintain a rather consistent performance when facing images other than the ones included in the addressed dataset.

While the constructed model presents a decent performance, it is hypothesized that it could potentially be possible to further increase accuracy and reduce the complexity of the considered features through the use of dimensionality reduction techniques. The implementation of principal component analysis (PCA) and linear discriminant analysis (LDA) into the model is therefore evaluated.

Principal component analysis (PCA) PCA first defines the hyper-plane that lies closest to the data, and then it projects the data onto it. The number of components to keep is set to a value of 0.95, which indicates the ratio of the

variance that we want to preserve. Its implementation reduces the accuracy of the model to a value of 70.26%; a significant drop of more than 5% that most likely does not compensate the simplification of features.

Linear discriminant analysis (LDA) In contrast to PCA, during training LDA learns the most discriminative axes between classes. These axes can then be used to define a hyper-plane onto which to project the data. The benefit of this approach is that the projection separates the classes as far apart as possible. The implementation of LDA brings the accuracy of the model up to 78.69%, which signifies an improvement of more than 3%.

2.4 SVM classifier

Support vector machines (SVM)[4] is considered as an alternative classifier to k-NN. This kind of classifier works completely different from k-NN, as it builds on the idea of separating classes with hyper-planes in a way that minimizes the amount of wrong instances and maximizes the margin between sides.

Taking into consideration the best parameters found for the version of the model using k-NN, only dense SIFT is this time considered, and the size of the codebook is fixed at 128. The carried out tests detailed in the following points focus on exploring the effects of using different scaled local descriptors for the dense SIFT, evaluating the normalization of said descriptors (with L2-norm and standard scaling), and exploring the use of different kinds of kernels for the SVM.

SVM with linear kernel Figure 8 displays the accuracy ratings for a SVM-based model with a default regularization term of $C = 1$ and a linear kernel. These ratings are showcased for different scale values of the dense SIFT local descriptors and for both the unnormalized and normalized descriptors.

There do not appear to be big differences in accuracy depending on whether normalization or standard scaling is applied or not. This small difference probably has to do with how all of the features in this particular case are already in a similar range that would not benefit from any further adjustments. The scale used for the dense SIFT does not seem to have a great influence on the performance either; only a very subtle increase in accuracy can be appreciated as scale grows.

Next, scale is fixed to a value of 6, and 5-fold cross validation in a grid search[5] is performed to optimize the regularization term C. This optimized term turns out to be of $C = 0.0001$, and using it brings the accuracy of the model (with standard scaling) from 77.45% to 80.42%.

SVM with radial basis function (RBF) kernel Figure 9 displays the accuracy of the SVM with a RBF kernel, a default regularization term of $C = 1$ and a kernel coefficient (*gamma*) set to "*scale*". Once again, ratings are portrayed for different dense SIFT local descriptor scale values and for unnormalized and normalized descriptors.

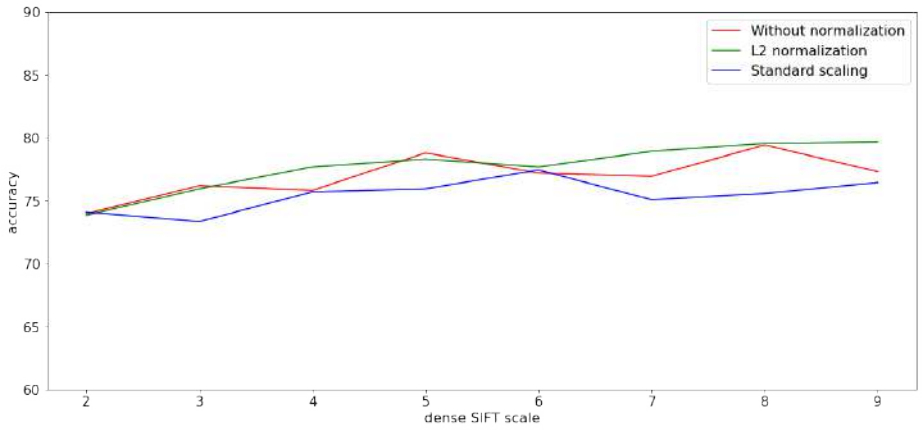

Fig. 8: Accuracy using default SVM with linear kernel for different descriptor scales and types of normalization.

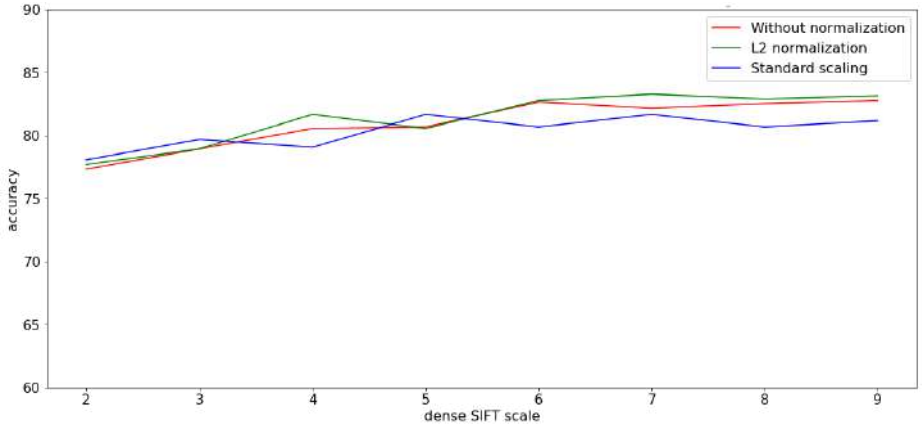

Fig. 9: Accuracy using default SVM with RBF kernel for different descriptor scales and types of normalization.

Once again, the SVM classifier with this new RBF kernel appears to deliver good accuracy out-of-the-box. Descriptors computed with a dense SIFT of scale 6 or higher seem to lead to the highest accuracy ratings. The application of normalization and standard scaling on the descriptors does not seem to lead to any significant changes in accuracy. There is a slight increase in performance as the scale considered in the dense SIFT grows, but this increase seems to stop at a scale of 6.

Similarly to the process done for the SVM with a linear kernel, 5-fold cross-validation and a grid search are performed to optimize the regularization and kernel coefficient *gamma* terms in an attempt to maximize the classifiers accu-

racy. This is done after having fixed the scale to 6 and having used standard scaling for the descriptors. The optimal parameters turn out to be the default ones: $C = 1$ and $gamma = scale$, leading to an accuracy of 80.67%.

SVM with histogram intersection kernel Finally, Figure 10 displays the accuracy ratings for a SVM-based model with a default regularization term of $C = 1$ and a histogram intersection kernel (determined through the expression in Equation 1, where A and B are two histograms). Ratings are again showcased for different scale values of the dense SIFT local descriptors and for both the unnormalized and normalized descriptors.

$$K_{int}(A, B) = \sum_{i=1}^{m} min\{a_i, b_i\} \tag{1}$$

The SVM classifier using a histogram intersection kernel yet again delivers a good accuracy without prior adjustments. This accuracy is actually the highest one so far, despite not having carried out an optimization of any parameters. This kernel however requires a significantly longer computation time compared to the prior ones. Normalization and standard scaling still do not seem to affect the classifiers accuracy much at all. Just as with the other kernels, there is a slight increase in performance as the scale grows, but this increase appears to stop at a scale of 6.

After once again fixing the scale at 6 and using standard scaling, 5-fold cross-validation and a grid search are performed to optimize the regularization term. The optimal value for this term turns out to be $C = 0.001$. The accuracy of

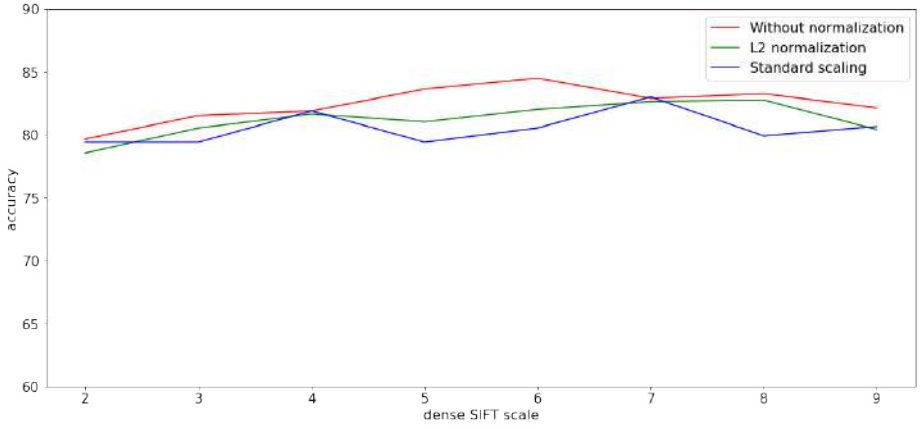

Fig. 10: Accuracy using default SVM with histogram intersection kernel for different descriptor scales and types of normalization.

the classifier is thus recalculated for this new adjusted C, presenting an updated value of 81.29% slightly higher than the 80.54% of the unoptimized model.

In an attempt to further increase the performance of the model, the implementation of spatial pyramids is considered next. This should in theory help the model take into account more concrete information regarding the approximate position certain elements or patterns usually appear in within images.

Spatial pyramids Until now every image has been assigned one word histogram. The use of spatial pyramids[6] is proposed to divide images into different segments and represent each of these with a histogram. This should provide additional information for each image, as location-related characteristics should now come into play.

The idea is to split images only into horizontal blocks (see second image in Figure 11) instead of splitting them both horizontally and vertically (as in the third image in Figure 11). The reasoning behind this is how landscapes tend to be more consistent in the horizontal direction. For instance, ground or water will always be covering the entirety of the width of an image's low portion. On the other hand, one cannot predict what the horizontal position of elements such as trees will be in a given image. That being said, images split into simple horizontal blocks should encompass nearly the same amount of information as images split in more complex grid-like patterns.

A function to compute the visual words of a set of training or test images following a spatial pyramid approach is built. This function is thus among other arguments provided with a number of levels to be considered, and iterates through each level performing the appropriate horizontal splits and computing histograms for each of these. Histograms from different levels are assigned particular importance through the use of weights.

The effects of using spatial pyramids is evaluated on an SVM classifier with a linear kernel built on dense SIFT descriptors with a scale of 6. The performance of the classifier for up to 3 levels is evaluated. The regularization term is optimized for each level using the 5-fold cross-validation and a grid search approach.

Fig. 11: General idea behind the use of horizontal blocks in spatial pyramids.

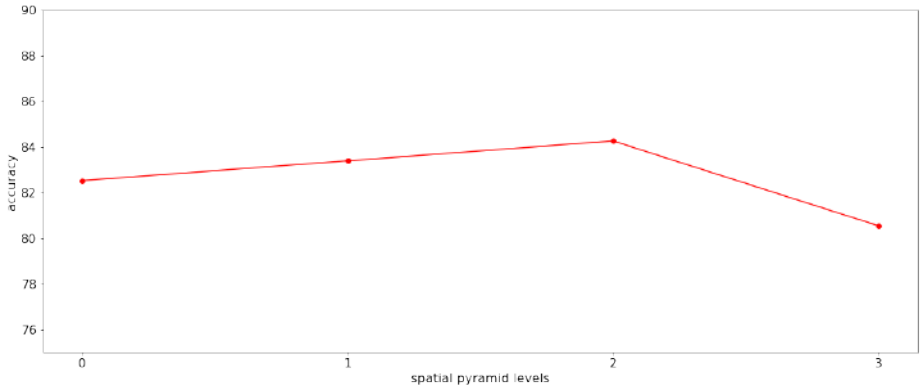

Fig. 12: Accuracy of SVM with linear kernel for different spatial pyramid levels.

As observed in Figure 12, spatial pyramids set to certain levels indeed appear to slightly increase the accuracy scores of the SVM classifier. A 2-level spatial pyramid leads to the best results (an accuracy of 84.26%), while at level 3 the performance starts decreasing.

The classification results for the best performing SVM-based BoVW model are showed in Figure 14. This model uses dense SIFT with a scale of 6px and a step of 5px, L2 normalization on the descriptors, a codebook size of 128, a histogram intersection kernel for the SVM, a regularization term of $C = 0.001$, and a level 2 spatial pyramid.

One of the most common cases of miss-classification is that of images from the "Opencountry" and "Coast" categories being confused. If the characteristics of these two categories are considered, it makes sense for these to get mixed up at times, as they both contain the images with the widest spaces, and tend to have many elements in common such as open skies, clouds, water, etc.

Fig. 13: Examples of correctly and incorrectly predicted images.

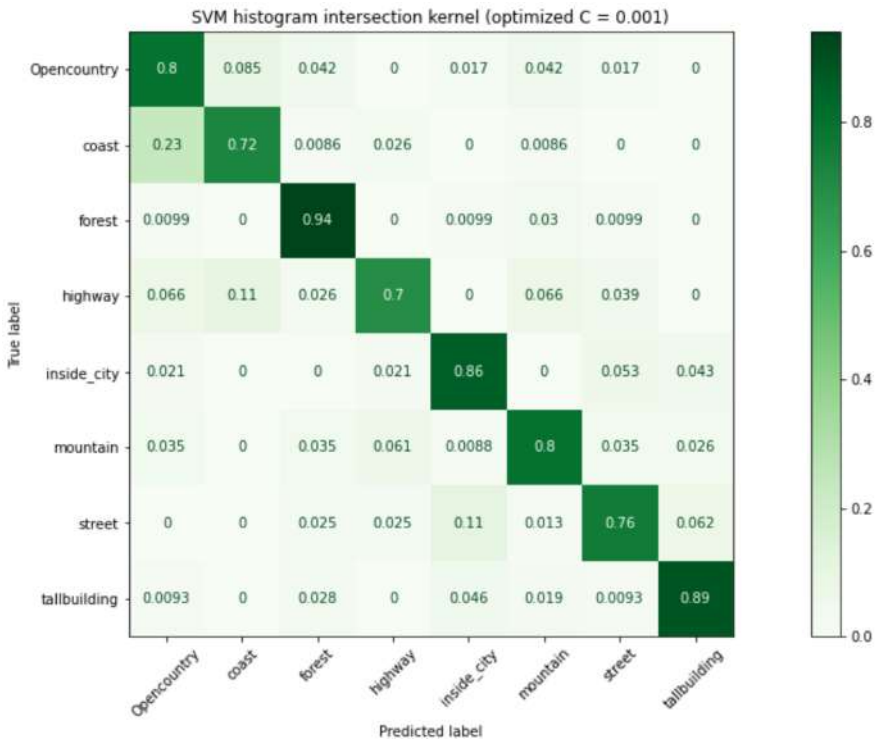

Fig. 14: Confusion matrix results of the best BoVW model.

3 Deep learning

Unlike the hand-crafted methods introduced so far, learning-based models like multi-layer perceptrons (MLPs) and convolutional neural networks (CNNs)[7] are able to construct their own features solely based on a set of training data through a series of iterative internal operations.

This section first introduces a simple MLP with just a few hidden layers to build a basic understanding of neural networks and how hyperparameters interact with the model. Then, it moves on to tackle the classification problem with more powerful CNN architectures using transfer learning. The section ends with the crafting of a CNN from scratch that is specific to the proposed dataset for optimized efficiency.

3.1 Multi-Layer Perceptron (MLP)

The MLP considered as a baseline is shown in Figure 15. It consists of an input layer followed by two fully connected layers that finally connect to an output layer. The model contains a total of 6,309,896 trainable parameters.

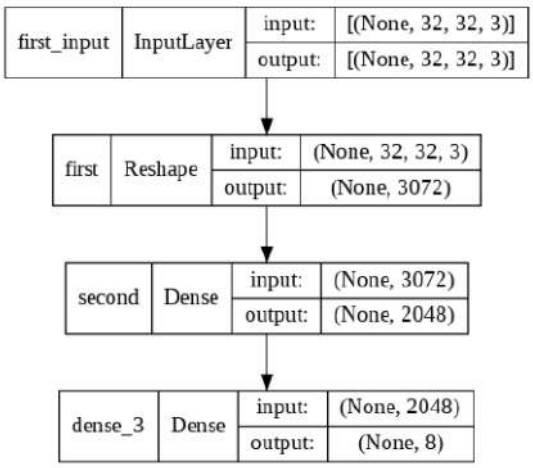

Fig. 15: Architecture of the baseline multi-layer perceptron.

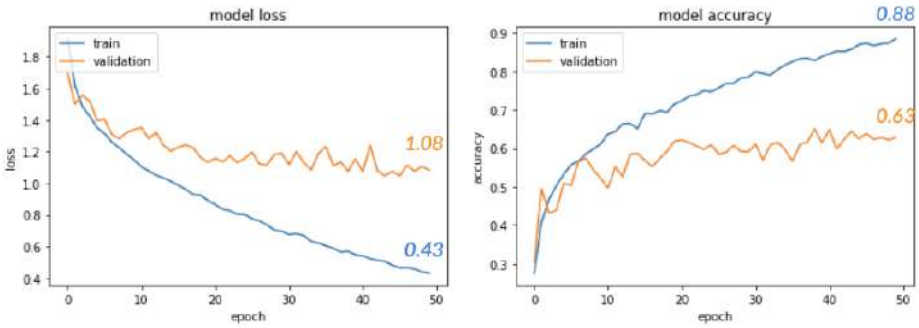

Fig. 16: Accuracy and loss evolution of the baseline MLP.

Looking at the loss and accuracy curve for training and validation data for the baseline MLP in Figure 16, an obvious case of over-fitting can be observed. After approximately 10 epochs, the performance for the training set keeps increasing while the one for the validation data practically stagnates. The validation accuracy is rather poor, but it shows a lot of potential for improvement.

Understanding network topology Experiments were conducted while setting certain hyperparameters (i.e. including image size, batch size, layer depth and number of nodes per dense layer) to different values. Despite how adjusting these hyperparameters individually is not the ideal way of carrying out the fine-tuning of the model, the purpose of these experiments is to evaluate the effect of each of these on the model's performance. All experiments are carried out with a limit of 50 epochs, while implementing an early stop on the validation loss with a patience of 5 epochs.

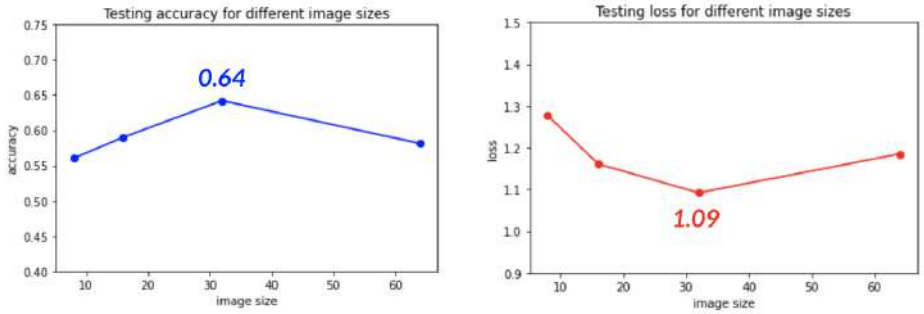

Fig. 17: Effect of image size on the baseline MLP.

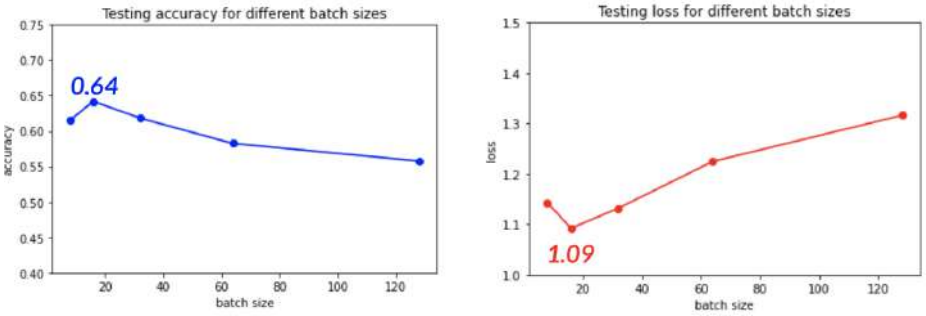

Fig. 18: Effect of batch size on the baseline MLP.

As observed in Figure 17, accuracy and loss values seem to become better up to image sizes of around 32x32px, after which they start getting worse again. The fact larger images do not necessarily work better has to do with how these are associated with significantly more parameters (e.g. an image size of 8 involves 411,656 parameters, while an image size of 64 involves 25,167,872). This leads to a higher extent of overfitting.

In terms of batch size, as it can be appreciated in Figure 18, accuracy appears to increase up to a batch size of approximately 16. Past this point, it progressively decreases. Larger batch sizes do not lead to overfitting within the established limit of 50 epochs. Each iteration however takes a longer time, and accuracy is not as good as for smaller batch sizes.

In order to test the effect of including several amounts of hidden layers in the model, the configurations portrayed in Figure 19 have been tested. As seen in Figure 20, the addition of more hidden layers does not appear to increase the models fitting performance, but rather the opposite. Maximum accuracy and minimum loss is achieved with only the two hidden layers in the original model, despite requiring significantly less trainable parameters than the other two setups. No major differences in terms of overfitting were observed between the models.

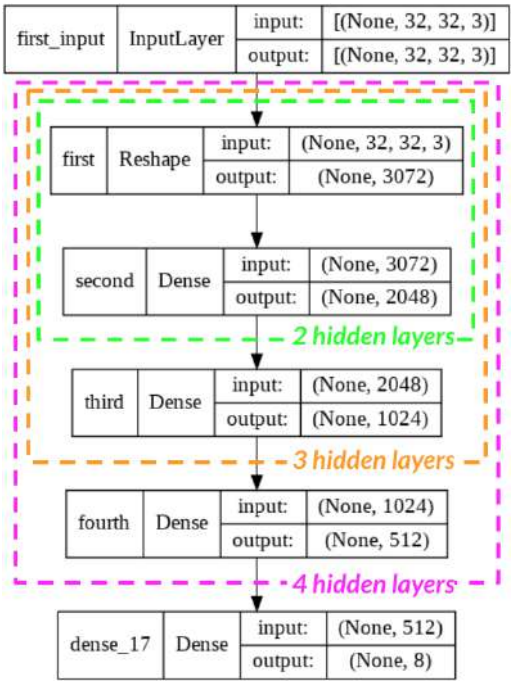

Fig. 19: Considered configurations of hidden layers.

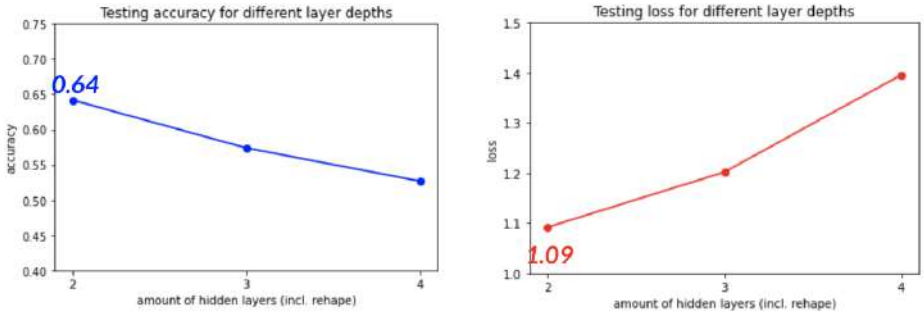

Fig. 20: Effect of number of hidden layers on the baseline MLP.

Next, different amounts of nodes for the output of the second layer of the baseline MLP are tested. Based on the results in Figure 21, the model's performance first seems to increase as more nodes are included in its dense layer. However, after approximately 2048 nodes, this progression stops and accuracy/loss remain almost constant despite the incorporation of more trainable parameters. This decrease in performance for higher numbers makes sense considering the consensus regarding how the nodes for the hidden layers of a MLP should decrease with depth.

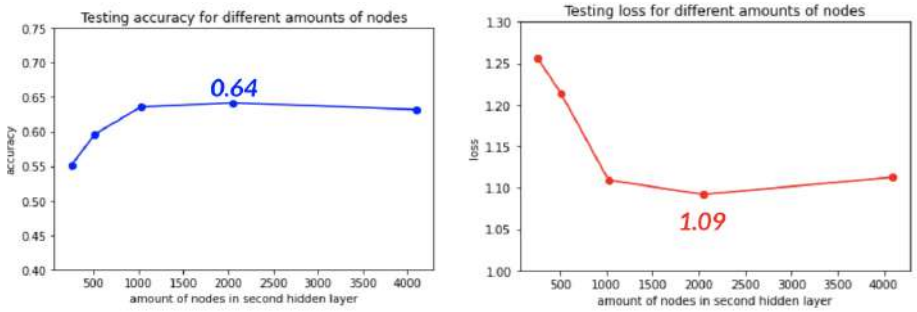

Fig. 21: Effect of number of nodes of the second layer on the baseline MLP.

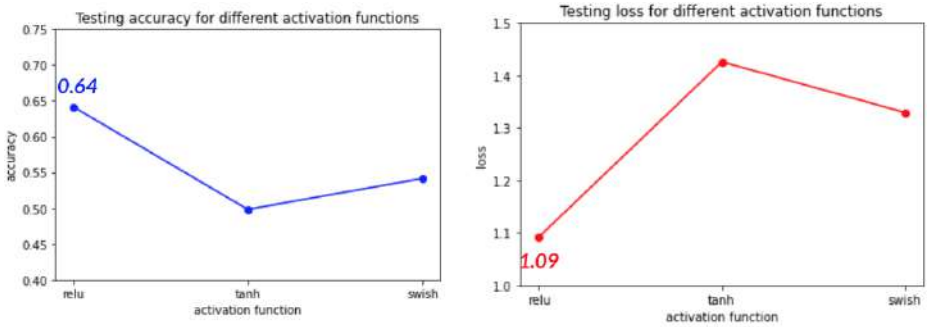

Fig. 22: Effect of activation function type on the baseline MLP.

Finally, different types of activation function for the fully connected layers are also evaluated. As displayed in Figure 22 The result shows that ReLU activation function leads to the best performing model overall, leading to approximately +10% more testing accuracy and +0.2 less loss than the other considered functions such as TanH and Swish. While these large differences between functions are strange, they might have to do with the irregularity of the testing loss curves for the TanH and Swish functions leading to an unexpected early stop.

Using learnt features A MLP can be cropped up to a certain layer, from where outputs can be extracted and used as features in other kinds of models[8]. Here a feature array can be extracted from the first and second layers of the baseline model as shown in Figure 23. These can then be used as descriptors for an SVM classifier[8]. As detailed in Table 2, the testing carried demonstrates how using the output of one of the deeper layers of an MLP for this approach can lead to a model with a very decent accuracy rating, surpassing that of the end-to-end model by more than 10%.

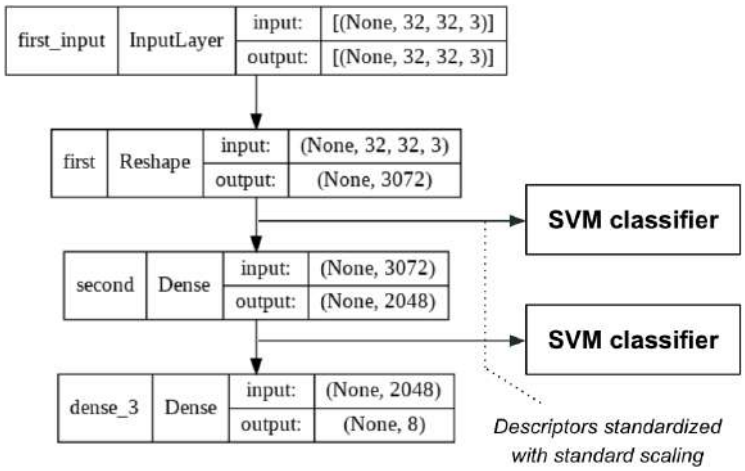

Fig. 23: Pipeline for using learnt features with SVM

Table 2: Validation accuracy for end-to-end model and MLP-SVM combinations.

Classifier model	Validation accuracy
End-to-end	64.19%
SVM from first layer output	58.10%
SVM from second layer output	75.49%

Patch-based approach The idea of dividing input images into multiple patches for training is here explored, which should help take advantage of the spatial information contained within said images. After the training process, the model makes a prediction for each patch, then it aggregates the predicted scores leading to a final prediction. The general structure of this patch-based approach is shown in Figure 24.

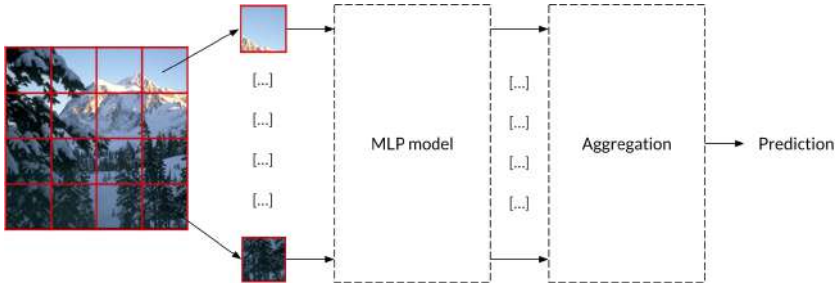

Fig. 24: MLP patch-based approach.

Several patch sizes are evaluated, leading to the results shown in Figure 25. Patch size indeed appears to have a considerable effect on the model's performance. Dividing images into 16 patches (size 64px) seems to lead to the highest accuracy, but other patch sizes also seem to improve the performance of the basic MLP. This has to do with how handling images through various patches helps provide some information to the model regarding common positioning patterns (e.g., skies, water, etc).

In addition, for further testing purposes, the outputs from the MLP patch-based model are extracted and used as descriptors for a BoVW that uses a SVM classifier. The validation accuracy of this alternative approach is once again evaluated for different patch sizes, as well as for various codebook sizes. It can be observed in the results in Figure 26 how the model's accuracy appears to increase with the size of the codebook. The increase in performance is however insignificant when compared to the required computation times after a size of around 256. Also, as expected, smaller patches once again provide the best results.

Overall, the patch-based MLP that makes use of BoVW presents slightly lower performance scores than the one using simple aggregation. This difference is of approximately 2% in validation accuracy, but it is still worth considering.

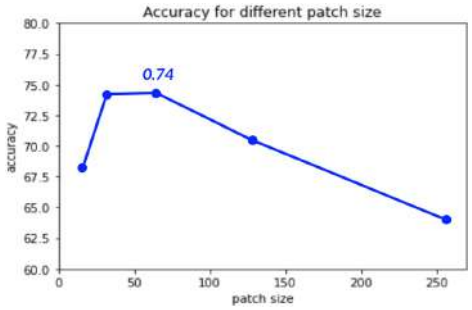

Fig. 25: Effect of patch size on the MLP.

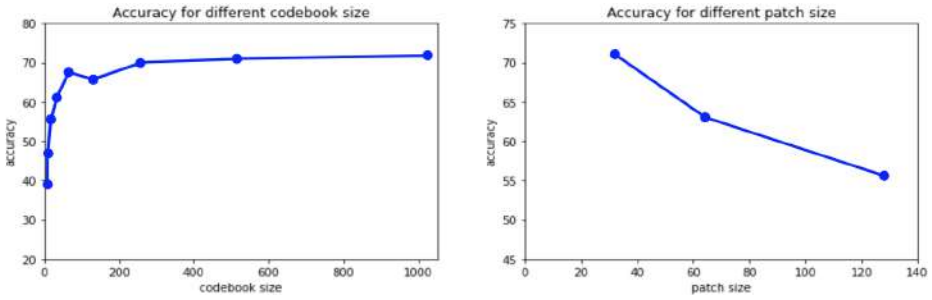

Fig. 26: Effect of codebook and patch sizes on MLP + BoVW model.

3.2 Fine tuning of pre-trained CNNs

The focus of this section is to leverage the power transfer learning through the use of pre-trained CNN models. Provided one of the models from *Keras Applications*, the idea is to progressively fine tune said model and later perform modifications to it so that a new model better tailored to the data at hand is achieved. The model used here is the *MoblileNetV2* architecture pre-trained with the *ImageNet* dataset. At only 3,538,984 parameters, it happens to be the most compact of the available models, and in turn one of the fastest ones.

The first step in adapting the given model to the application at hand is to change its final classification output to suit the 8 classes that images are to be classified in. It is also necessary to pre-process images to suit the input format of the selected model. This is done according to the documentation, that mentions how *MobileNetV2* is expecting normalized values in the range [-1,1] centered at 0 [9]. The model is then re-trained to suit the image data addressed in this project.

As displayed in Figure 27, this simple out-of-the-box setup shows a validation accuracy of up to 95%, significantly outperforming any of the previous attempts. The validation accuracy curve remains constant after an initial increase in the first few epochs. This shows the pre-trained model is already adequate to extract all relevant features for describing the problem at hand, and only a few epochs are needed to adjust the classifier.

For the sake of making the classification problem at hand more challenging, the original dataset of 1881 images is reduced to one of only 400 images. This immediately lowers the validation accuracy of the same model by approximately 2%, which comes to show the importance of the amounts of available data. Several strategies are implemented to try to increase this accuracy rating and lower the amount of parameters of the model.

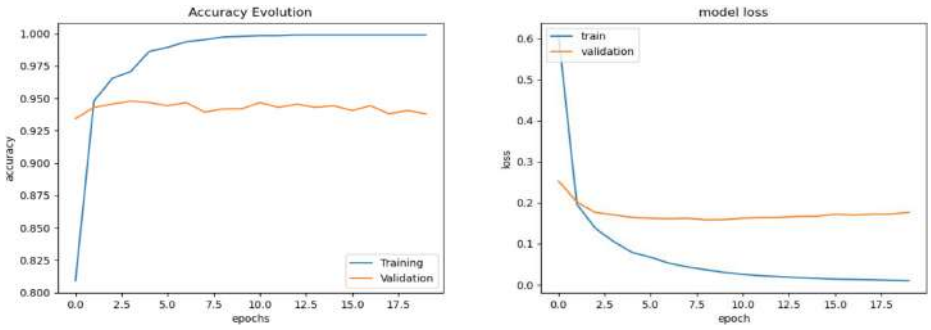

Fig. 27: Accuracy and loss evolution of the re-trained *MobileNetV2*.

Data augmentation Data argumentation is proposed as a way to artificially inflate the amount of training data available to the network. The following alterations are considered, individually and combined:

- Random zooming up to 40%
- Horizontal flipping
- Random rotation of up to 1° in any direction
- Random increase/decrease in contrast of up to 40%

As displayed in Table 3, most kinds of data augmentation do not seem to lead to any improvement at all in the model's classification performance. The only case that shows slightly better results is that regarding the use of the random rotation alone (approximately 0.5% higher accuracy), but it is unclear if this could have to do with the random nature of the tests.

This lack of effectiveness is most likely related to the fact the images from the dataset are professionally taken (without obvious rotation, colour distortion, etc.). Therefore, the augmentation pre-processing shows no real improvement and is possibly guilty of introducing noise to the input data.

Fig. 28: Examples of altered images in data augmentation.

Table 3: Validation accuracy using different kinds of data augmentation.

Individual alterations		**Combined alterations**	
Alteration	**Val. accuracy**	**Alteration**	**Val. accuracy**
Horizontal flip (F)	92.48%	F + R	92.83%
Rotation (R)	93.53%	R + Z	90.34%
Zoom (Z)	93.00%	Z + C	91.69%
Contrast (C)	92.83%	F + R + Z + C	92.22%

Cropping the model In an attempt to further simplify the used architecture, the CNN is cropped to several layer depths and the output of these is directly connected to a global average pooling and final softmax layer. The considered levels for the tested crops are indicated in Figure 29, and are based on the layer structure from the original article for the *MobileNetV2* [9].

As it can be observed in Table 4, while none of the crops improve accuracy scores, using the output of the deepest layer level (H) only lowers it by less than 0.5% while reducing the number of parameters by more than 422,000. Cropping the CNN to earlier layer levels thus appears to have certain potential in terms of accuracy/parameter trade-off. It would however perhaps be good to completely re-train each of the cropped versions of the model to further get the best performance out of these.

Adjusting hyperparameters A hyperparameter search[5] is finally carried out in an attempt to find the optimal adjustments for aspects like the optimizer type, the learning rate, the number of epochs and the poling method to be used. This search provides the results detailed in Table 5.

The performance achieved by using these adjustments is however not substantially better than the one that is already being achieved. It generally appears to be rather difficult to improve the *MobileNetV2* model in terms of accuracy

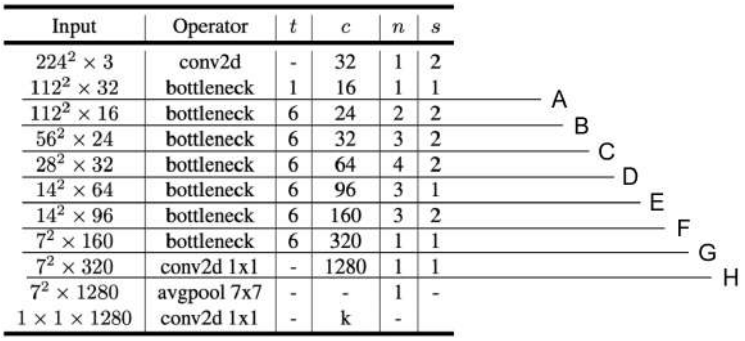

Fig. 29: Considered levels for the cropping of *MobileNetV2*.

Table 4: Accuracy ratings and parameter counts for different layer crops.

	A	**B**	**C**	**D**
Val. accuracy	39.55%	56.03%	52.36%	69.36%
Parameters	2,120	7,752	17,208	59,272
	E	**F**	**G**	**H**
Val. accuracy	75.09%	86.36%	87.11%	91.56%
Parameters	249,288	559,432	1,366,152	1,845,832

Table 5: Accuracy ratings and parameter counts for different layer crops.

Parameter	Search space	Result
Optimizer	sgd, rmsprop, adam, adadelta, adagrad	adam
Learning Rate	1e-4 - 1e-2	4.84e-3
Pooling method	max pooling, average pooling	average pooling
Epochs	2, 6, 17, 50	50

and parameter count. This model has already been devised to be compact due to its intended suitability for mobile devices, and it is immediately quite effective on the simple image classification case at hand. The introduction of more organic data is the only aspect that seems to considerably increase the model's performance.

3.3 Creating a CNN from scratch

Having been able to demonstrate the potential of CNNs by working on an existing model, the idea in now to construct a new model from scratch that is as tailored as possible to the image data being addressed. The most important considerations to build this model are the following:

- **Good classification performance**. The model should be able to correctly classify as many images as possible.
- **Compactness**. The number of parameters the model is comprised of should be kept to a minimum.

Both of these aspects are to be taken into consideration through the use of the expression in Equation 2 as a performance metric.

$$performance\ ratio = \frac{accuracy}{number\ of\ parameters/100K} \tag{2}$$

An initial baseline network based on the simplistic structure of the *AlexNet* [10] is set up as a starting point. This network is basically comprised of repeating convolution blocks (2 convolutions & max pooling) followed by a fully connected layer structure. No regularization elements are however implemented yet. This model yields the performance displayed in Figure 30, where a clear case of overfitting can be appreciated. This rather poor performance makes sense considering the excessive amount of parameters of the model for the problem at hand and the lack of any regularization.

With the baseline model in place, a series of steps are taken to progressively improve it. These are summarized in Table 6, which showcases the progression of the model's performance, the number of parameters, and the consequent performance ratio measure.

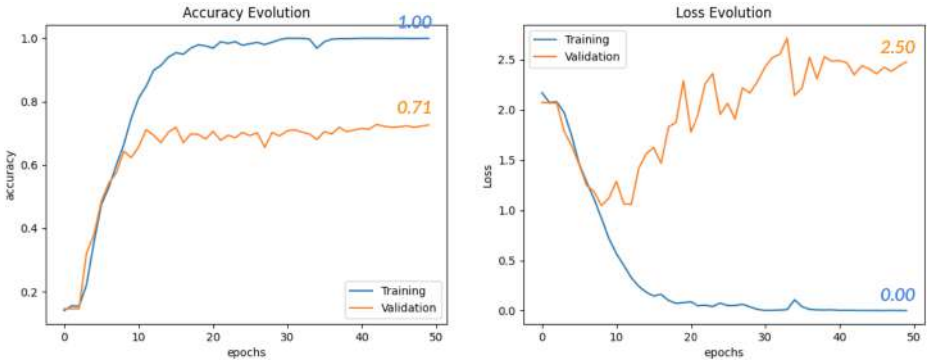

Fig. 30: Accuracy and loss evolution of the baseline model.

Table 6: Summary of CNN building process.

Step	Accuracy	Loss	Parameters	Perf. ratio
Baseline model	71.30%	1.6	34,679,496	0.0021
Without fully connected layers	60.91%	3.14	1,640,456	0.0371
Single Conv2D in repeat. blocks	62.60%	2.69	902,792	0.0727
Dropout (0.3)	67.95%	4.37	902,792	0.0809
Initial Batch Norm	73.92%	1.57	902,804	0.0819
Intermediate Layer Norm	75.40%	2.37	903,188	0.0770
Global Average Pooling	88.78%	0.43	380,948	0.2263

Some of the tests in prior sections showed how adding multiple fully connected layers towards the end of the model does not improve results much, but it increases the number of parameters a great deal. With this in mind, only a softmax layer for the classification part of the model is maintained, which reduces the number of parameters by almost 30 times ($34M$ to $1.6M$). The parameter count is however still very high, so as a next step the basic convolution structures of the baseline model are reduced from two to just one convolution. This brings down the number of parameters to around $900K$; a significant improvement in compactness.

The next priority is to further reduce the overfitting issues by adding regularization elements to the system. First, dropout is added before the softmax dense layer, where the largest amount of parameters are located. The improvement in accuracy is noticeable (approximately a 5% increase), which is proof of the reduction in overfitting. Second, batch normalization is added before the first convolution to normalize the data, which makes the process more straightforward for the network. Third, to complement the batch normalization, a layer normalization is added before the intermediate layers. Since the training is done with small batches it is more suitable to use layer normalization, as batch normaliza-

tion adds bias assuming a mean and standard deviation from a small number of images. Finally, the max pooling at the end of the CNN is substituted by a global average pooling, which results in a huge increase in performance of more than 13% and a significant reduction in parameters.

Having reached a relatively reasonably accurate and compact model with a performance ratio of 0.226, several other parameter adjustments are evaluated and the implementation of certain techniques with the potential of further improving the model is considered.

Influence of image size Several image sizes are considered for the input in an effort to determine how this affects the classification performance of the system and the training times. The results of these tests are displayed in Figure 31. It can be observed how performance increases for larger image sizes, but so does the training times. If comparing for instance the trade-off between the small increase in performance when going from an image size of 128x128 px to one of 256x256 px and the large increase in training time, it becomes apparent that it might not be worth going beyond a size of 128x128 px.

Depthwise separable convolutions Another approach to further try to decrease the number of parameters of the model is here considered. It involves the implementation of depthwise separable convolutions (DSC) that can be used to simplify the existing convolutions into simpler operations that require less parameters. As pictured in Figure 32, the idea is to perform the action of a standard kxkxM kernel through a simpler depthwise kernel (kxkx1) followed by a pointwise kernel (1x1xM).

The implementation of depthwise separable convolutions turns out to not only maintain the classification accuracy of the model, but to increase it by around 2% by making the model more generalized. The parameter count is brought from 380,000+ parameters to less than 55,000, which is clearly a huge improvement. This is reflected in an updated performance ration of 1.667.

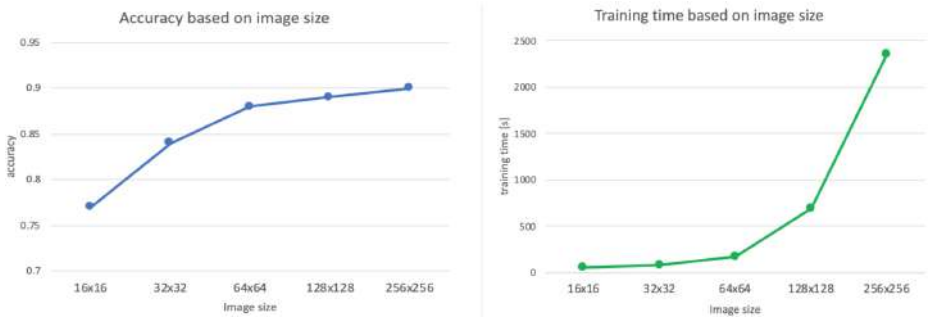

Fig. 31: Accuracy and training time evolution for different image sizes.

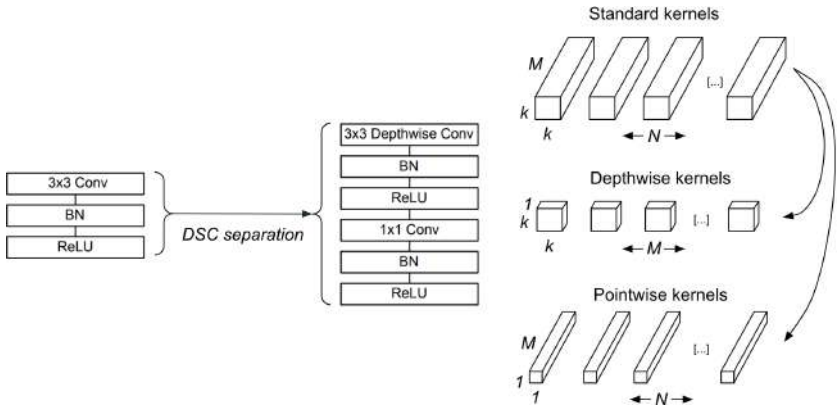

Fig. 32: General concept of the depthwise separable convolutions.

Data augmentation Once again, data augmentation is proposed as a way of providing the model with more information in order to boost its learning. The same kind of image alterations as in Section 3.2 are here implemented, and in a similar fashion to the results obtained in that same section, there does not seem to be any significant improvements in accuracy for any of the considered data augmentation cases.

Class weights Despite the fact the available image data sets are quite balanced in terms of how many images are included in each category, the implementation of class weights is still evaluated. As detailed in Figure 33, weights set proportionally to the percentage of available data for each category. While this should in principle help compensate any bias from data invariance, no increase in model performance is appreciated in this case.

Adjusting optimizer and learning rate Similarly to the hyperparameter tuning carried out in Section 3.2, a search is carried out to find out which optimizer and learning rate settings could maximize the performance of the model. The resulting optimizer turns out to be *RMSprop* and the optimal learning rate has a value of 0.00203. However, using these does not appear to lead to any significant improvements in the model's performance either.

Fig. 33: Implementation of class weights.

After having carried out all of the detailed steps, the resulting CNN model displays the structure in the Figure 34 below.

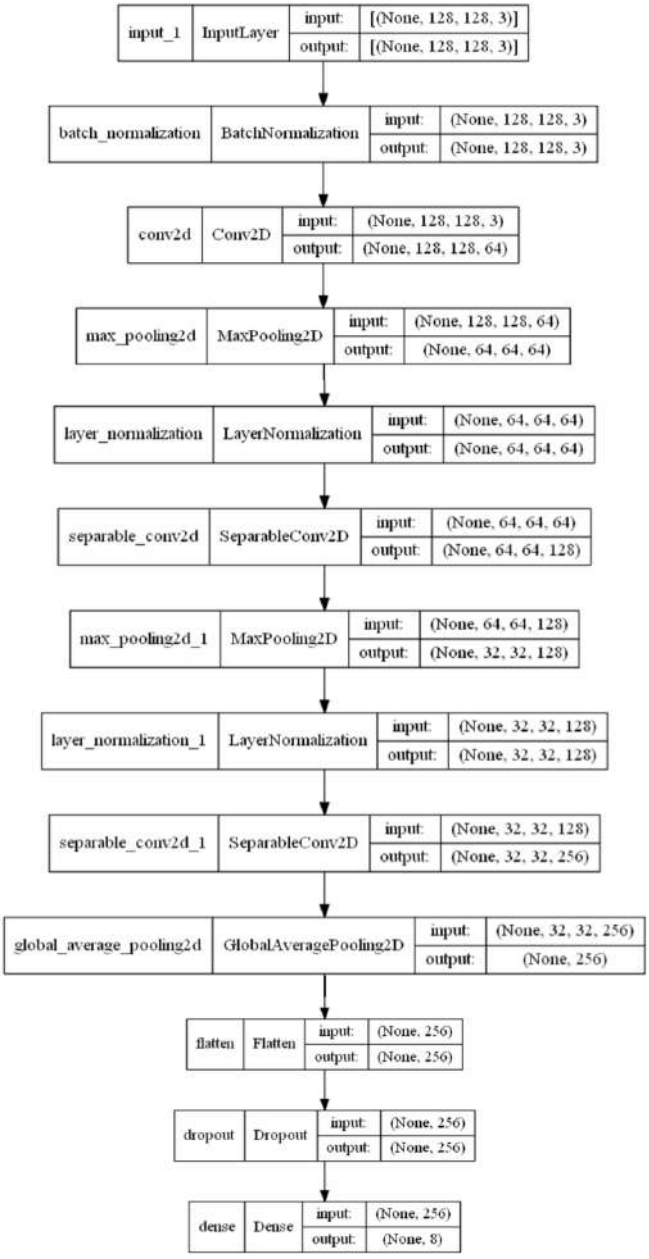

Fig. 34: Final CNN structure.

Figure 35 shows the general performance of this final model. It is noticeable how both training and validation curves for accuracy and loss are much smoother and closer to each other, which indicates that the overfitting issue has been solved.

Figure 36 also displays some of the wrong predictions carried out by this final classifier. While some categories are almost always correctly handled, there is still some confusion going on between images from classes such as "Opencountry" and "Coast", which most likely has to do with the same reasons as in previous sections. These are confusions that even us as humans would struggle with, so it makes sense for the model to have the same problem.

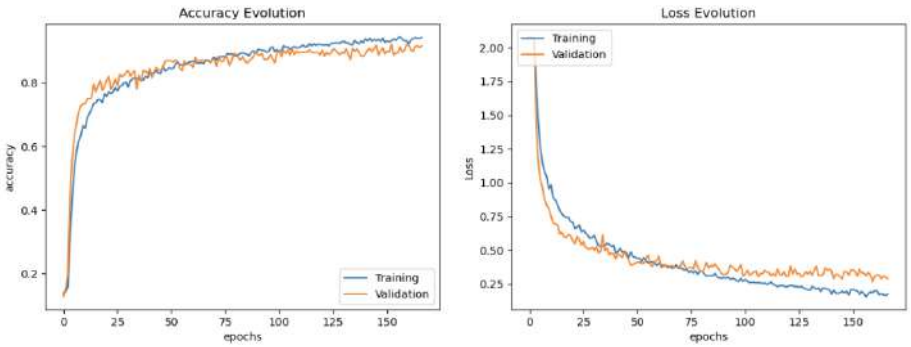

Fig. 35: Accuracy and loss evaluation curve of the final classifier.

Fig. 36: Table showing wrong predictions made by the final classifier.

Conclusions

The use of local descriptors as hand-crafted features through the implementation of a Bag of Visual Words approach is initially explored. They are easy to understand relative to the deep learning methods, but models using these have important limitations in terms of performance.

Next, a series of learning-based models are constructed and evaluated as additional alternatives. Pure MLPs use many parameters, are prone to overfitting, and are not translation invariant, which makes them a poor choice for image classification. Convolutional neural networks are much more effective tools than MLPs in image classification problems like the one at hand. Through the use of kernel-based operations, they introduce scale and position invariance in regard to recognizable elements in the images. Using sparse connections instead of full connections like the ones present in MLPs also make them reasonable in terms of parameter counts if designed properly.

The key to image classification is designing a system suitable for the size and traits of the data at hand, implement proper regularization techniques, and perfect the model through testing.

References

1. F. Perronnin, C. Dance, G. Csurka, M. Bressan. Adapted Vocabularies for Generic Visual Categorization,ECCV
2. Lowe, David G. "Object recognition from local scale-invariant features." Proceedings of the seventh IEEE international conference on computer vision. Vol. 2. Ieee, 1999.
3. Lowe, David G. "Distinctive image features from scale-invariant keypoints." International journal of computer vision 60.2 (2004): 91-110.
4. Csurka, Gabriella, et al. "Visual categorization with bags of keypoints." Workshop on statistical learning in computer vision, ECCV. Vol. 1. No. 1-22. 2004.
5. Bergstra, James, and Yoshua Bengio. "Random search for hyper-parameter optimization." Journal of machine learning research 13.2 (2012).
6. Lazebnik, Svetlana, Cordelia Schmid, and Jean Ponce. "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories." 2006 IEEE computer society conference on computer vision and pattern recognition (CVPR'06). Vol. 2. IEEE, 2006.
7. Chatfield, Ken, et al. "Return of the devil in the details: Delving deep into convolutional nets." arXiv preprint arXiv:1405.3531 (2014).
8. Cimpoi, Mircea, Subhransu Maji, and Andrea Vedaldi. "Deep filter banks for texture recognition and segmentation." Proceedings of the IEEE conference on computer vision and pattern recognition . 2015.
9. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L. C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4510-4520).
10. Krizhevsky, Alex; Sutskever, Ilya (2017). "ImageNet classification with deep convolutional neural networks"