Animal Classification

Gonzalo Pedregosa Ordóñez
Garben Tanghe
Thibo Vanquaethem
Marie Vanzieleghem

December 2, 2018

Abstract

This is the report for the group phase of the project in the context of the course Machine Learning, given in the Master of Science in Computer Science Engineering. Most of the information in this report originates from the Keras, scikit-learn and TensorFlow documentations. Other sources are references at the end.

1 Problem

The project considers the problem of animal classification with the help of machine learning or deep learning. The goal is to create and train a model that can classify 11 species with a cross-entropy error as small as possible. The 11 species are: bobcats, chihuahuas, collies, dalmatians, German shepherds, leopards, lions, Persian cats, Siamese cats, tigers and wolves.

The problem is of the type supervised classification, the desired output of the train set is available and it is necessary to predict probabilities. The cost function that will be used is cross-entropy error (also called logistic loss or log loss), as mentioned above.

2 Data Analysis and Preprocessing

2.1 Data Analysis

The colored images are scraped from Google. That's the reason they have different sizes and shapes. The image width and height range from around 200 to 1920 pixels. The aspect ratio varies for both landscape and portrait photos.

The animals are not always situated in the center of the image and they do not always have the same size. They can be laying down, running, looking in to the camera or the image can be taken from their side. Sometimes animals are partially covered by trees, blankets, etc.

There are some outliers in the train set. Some images have the wrong label, for example bobcat_0080.jpg is in fact an image of a pheasant and german_shepherd_0027.jpg portraits a Border collie. Some images contain humans with their pet (e.g. chihuahua_0006.jpg). Others contain more than one animal of the same class (e.g. tiger_0018.jpg) or even picture animals of different classes (e.g. german_shepherd_0105.jpg). A combination of these is also possible: german_shepherd_0034.jpg contains one German shepherd, one collie one other dog species and 7 people.

The images in the test set have the same variation in size and aspect ratio. They also have the same types of outliers (people, more than one animal of the same species, different animal species in one image, etc.). Most of the time, the animal is situated near the center of the image and not covered by obstacles.

It is important to note that there is class imbalance. There are 517 pictures of German shepherds, meanwhile there are only 55 photos of wolves. A histogram of the animals in each class can be found in figure 1. To cope with this the class weights are automatically adjusted inversely proportional to the class frequencies of the input by the models. Also to make sure the class imbalance is correctly represented during data splits in batches for training/validating/testing, stratified K-folds are used when performing grid searched cross validations, but also when initially splitting $25\,\%$ test data from the train data with train_test_split by setting parameter stratify to train_labels.

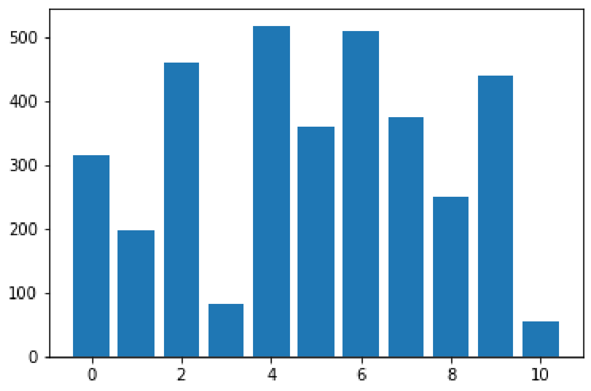


Figure 1: Histogram of the train labels.

To improve the ratio of important information on an image, background removal was tried. Our very simple method relied on creating a mask for the original image that covered as much background as possible (figure 2), giving this as parameter *'mask'* to the OpenCV methods *'detectAndCompute'* and *'goodFeaturesToTrack'* when extracting image patches with different feature descriptors. The mask was created by applying a threshold on the a*-axis from the CIELAB colour space, which describes the green-red components in the image. This threshold was based on the observation that most animals to-be classified are wild-life animals, therefore most background will consist of green or muddy brown terrain, while the animals themselves do not.

However, this naive observation was questioned fairly quickly, also combined with the lacking or very limited model performance gain, which could well-be caused by the randomness in data splitting. One could argue that most wild-life animals are evolutionary adopted to their environment, acting as a camouflage mechanism. Also, some

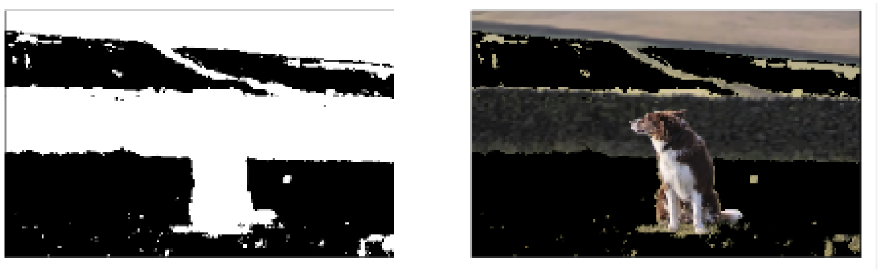


Figure 2: Background mask of an image.

animals have green eyes, causing parts of animals to be masked. This makes the threshold for background subtraction in colour space hard to pinpoint. Some images even had green filters applied to them, like persian_cat_0307.jpg. This raised questions on whether it is desirable to make our model colour dependent anymore, since trying to optimize this is severely hindered by slow I/O-operations.

Finally the preprocessing step of background removal was dropped, based on following observation on the influence of background features [1]. For many images background is not completely uncorrelated from the foreground and may provide *'hints'* for recognition tasks. It is easy to imagine that e.g. cars can easily be recognized from other objects, since they mostly appear with roads or parking lots in the background. This observation can be extended for our classification task. Most bobcat images contain forest in their background, while lions have grasslands, stones and dirt in their backgrounds and the cats and dogs mostly familiar home-like environments. It could well-be that the model uses this to make it's predictions more discriminatory.

As a final thought, another possible preprocessing method to improve the ratio of foreground features could be using existing object detection frameworks that can recognize the presence of an animal in the image and draw a box around it. Zooming the images on the detected box should expectedly increase the ratio of object features, increasing their importance in the classification task and only using the minority of background features for discriminatory means. However this is pure speculation and not tested on model performance gain, since the idea only came to mind moments before the deadline.

Data augmentation was only used for the convolutional neural network (CNN) and will therefore be discussed in section 5.4.

2.2 Feature Extraction and Analysis

Features are extracted from the preprocessed images using feature extractors like Scale-Invariant Feature Transform (SIFT) and Speeded Up Robust Features (SURF). Afterwards, a **Bag of Visual Words** (BOVW) is created. This concept originates from Natural Language Processing (NLP), which uses Bag of Words (BOW). The image features in a BOVW are comparable to words in a BOW. A BOVW is created as follows:

- Detect point-of interests using existing feature extractors.
- Cut out small, square areas (patches) around these points.

- Apply unsupervised clustering across all extracted regions in train data and determine cluster means. The amount of clusters is also called dictionary size, or codebook size.
- Extract points of interest for each image (train or test) and assign them to most similar cluster. Also count how many times each cluster was found in the image, this is the frequency of the feature.

SURF performed best, but takes a lot of time to execute. SIFT is therefore preferred and also most used in literature. A combination of multiple feature extractors is even better and therefore used to train the final models. Since SIFT and SURF are rotation and scale invariant, those augmentations won't have much effect on the model.

The unsupervised clustering step based on K-means, introduced another hyperparameter of dictionary size. Too many clusters, resulted in overfitting the train data too much, while too few clusters simply underfits the training data.

For SIFT and a simple linear model a codebook size of 480 performed best. This amount of clusters was extensively optimized by grid search in the previous phase. For each of the feature extractors, a codebook is now created. The codebook_size parameter is each time set to 400, which proved to be a sufficient sub-optimized dictionary size across different feature extractors. Full, greedy optimization of the full set of dictionary sizes turned out to be too computationally complex. Afterwards, all images (train or test) are encoded using this codebook and the encoded features are then concatenated to obtain one big feature vector or fused feature descriptor formed by late fusion, as stated in literature [2].

Features can also be encoded in another way. The two main criticisms or limitations of the BOVW-model are it's ignorance towards spatial relationships among patches, which are of key importance in image representations, and the fact only (robust) linear statistics are being used. Based on the second observation, comparative studies have been made on different classification pipelines [2]. By encoding first and second order statistics, the classification accuracy could be improved significantly over the standard BOVW-model, while also decreasing the codebook size and therefore the computational effort.

$$u_{jk} = \frac{1}{N\sqrt{\pi_k}} \sum_{i=1}^{N} q_{ik} \frac{x_{ji} - \mu_{jk}}{\sigma_{jk}},$$
$$v_{jk} = \frac{1}{N\sqrt{2\pi_k}} \sum_{i=1}^{N} q_{ik} \left[\left(\frac{x_{ji} - \mu_{jk}}{\sigma_{jk}} \right)^2 - 1 \right]$$

Figure 3: Mean and covariance deviation vectors.

One of these encoding schemes is based on **Fisher Vectors** (FVs), being a special case of the more general Fisher Kernel. The scheme can be understood by revisiting the last two steps of the BOVW-creation. Instead of using K-means unsupervised clustering, a (unsupervised) *Gaussian Mixture Model* (GMM) is applied, which assumes all data points are generated from a mixture of Gaussian distributions. The model tries to fit the training data by tweaking the distribution parameters. The amount of mixture components, or modes, is now the dictionary size and is usually significantly smaller compared to other dictionaries. In the encoding step, the GMM associates each local feature descriptor to a mode in the mixture with a strength, being the posterior probability. By utilizing these probabilities, the mean and co-variance deviation vectors (figure 3) can be calculated for each mode. Stacking these vectors results in the FV encoding, or global signature, for each image. On their turn, these vectors can then also be used, after some pooling method, for a classification task. The encoding scheme resulted in improved model performance in most cases.

Similar as the previous BOVW scheme, the unsupervised clustering step based on GMM also introduces another hyperparameter, namely the number of mixture components or modes. Same observations apply, too many modes results in overfitting the data, while too few underfits and does not represent the data well. This hyperparamater was optimized with a grid search. The optimal number of modes being 7, so all codebooks for different feature extractors are created with 7 mixture components.

Then, the train set is split into a train set and a validation set. train_test_split does a random split, so it shuffles the data before splitting. The fraction of the images split of to the test set is $25\,\%$. To take class imbalance into account, the stratify parameter is set to train_labels. This split is performed before feature selection and model optimization, and used for model evaluation. Because this split is done after codebook training, there is some data leakage during training. For the final model, no data from the test set is leaked.

Now, feature normalization is added, since many estimators are designed with the assumption that each feature takes values close to zero or more importantly, that all features vary on comparable scales. The normalization is done by using a Normalizer as the first step of the pipeline.

Finally, a label encoder is used to obtain numerical labels (0: bobcat, 1: chihuahua, ..., 10: wolf).

2.3 Analysis of Feature Importances

Principal Component Analysis (PCA) is used to reduce dimensionality. PCA uses Single Value Decomposition (SVD) to project onto a lower dimensional space. Since PCA is sensitive to scaling, the features were normalized first, as previously mentioned. The plot in figure 4 shows that about 270 features explain at least $60\,\%$ of the variance. KernelPCA outperforms PCA, but it does not return the proportion of variance explained, so a plot for that is missing.

Linear Discriminant Analysis (LDA) only returns 11 dimensions, which causes severe overfitting, because the model is not complex enough.

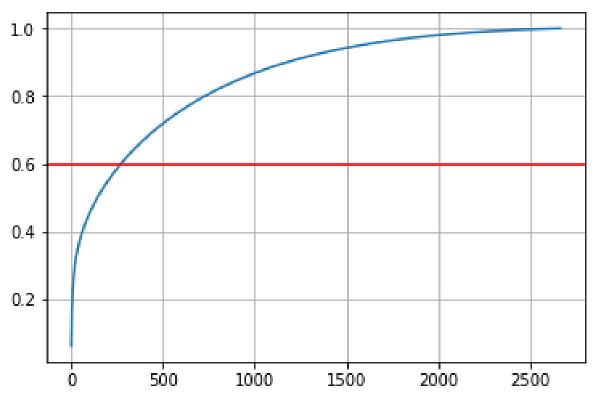


Figure 4: Cumulative sum of PCA eigenvalues.

3 Machine Learning Approach

3.1 Model

For each model in section 5, the model and its parameters used will be explained first. Its hyperparameters are then optimized, while training on the train set, that was split off earlier. The final model is then trained again on the whole train set and probabilities are predicted for the test set, so they can be written to a CSV submission file.

3.2 Validation Strategy

In order to make a decision for a model, it must be possible to compare models. A comparison is made between the validation log loss, validation accuracy, train accuracy and 5-fold cross-validation accuracy. Validation log loss is considered as the most important one, since it is that metric that needs to be optimized. Models account for class imbalance by using stratified K-folds for grid search and cross-validation.

3.3 Hyperparameter Optimization

To tune hyperparameters, GridSearchCV with 5 folds is used. The parameters of the estimator are optimized by cross-validated grid-search over a parameter grid (here tuned_parameters). The scoring parameter is set to ‘neg_log_loss’ since that is the metric that needs to be optimized. The best set of parameters is then used to return the best model found. The exact optimizations for each model will be explained in section 5.

3.4 Supervised Feature Set Optimization

SelectKBest and RFECV were tried out to minimize the amount of features. Selecting a subset of the features with SelectKBest did not improve the scores. RFECV did slightly improve the model, but the training speed increased a lot.

4 Model Evaluation and Error Analysis

4.1 Analysis of Over- and Underfitting

An overview of all scores (train scores, validation scores and submission score) can be found in table 3 near the end of the report.

4.2 Analysis of Errors

The final score is computed using log loss (cross-entropy error), this error measure requires the classifier to be very confident.

For each model, the confusion matrix and the normalized confusion matrix are drawn.

5 Models

5.1 Linear Models

The linear model that was used, is a support vector machine (SVM) or a support vector classifier (SVC) to be more precise. To make the model linear, a linear kernel must be used (kernel='linear').

The only hyperparameter of this model can be optimized now, being the penalty parameter C of the error term. At first, C is optimized using 8 values in a logarithmic scale between 1.0×10^{-4} and 1.0×10^{3}. We can see that an optimum must lay in the interval between 1.0 and 10.0, so we scale down our search interval around these values and eventually decide on 2.0 as the most optimal value in the case of FV encoding and 5.0 in case of BOVW.

As previously mentioned, the SVC model is combined with a kernelPCA and Normalizer. The PCA components were determined in an ad-hoc fashion based on the portion of variance explained (figure 4) and the elbow of the curve. When adding more components that explain larger portions of variance, the model starts to overfit, without gaining performance in validating score.

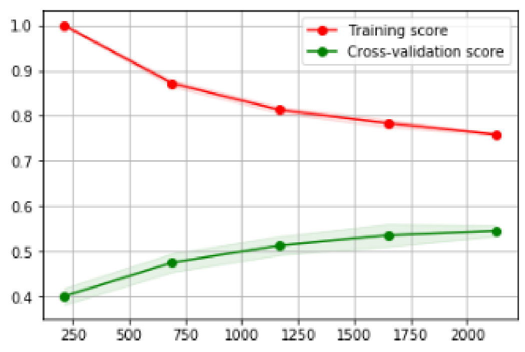
The linear model uses a fused global descriptor of following feature extractors: [SIFT, FREAK, BOOST, DESC, DAISY, LUCID, ORB] with each dictionary size or modes set at 400 (BOVW) or 7 (FV), since they gave the best overall performance out of any other combination. For convenience, these feature vectors are pickled and used in further (non-linear) models.

The completely optimized model with BOVW-encoded feature descriptors has a log loss of 1.26, an accuracy on the train set of $75.1\,\%$, an accuracy on the test data of 55.5 %, cross-validation accuracy 53.7 % and cross-validation standard deviation 2.10 %. The submissions gets a score of about 1.22.

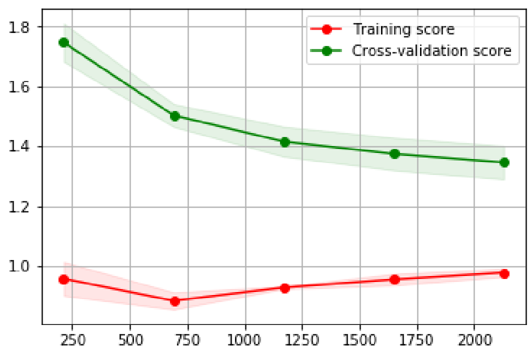
The completely optimized model with FV-encoded feature descriptors has a log loss of 1.20, an accuracy on the train set of 74.3 %, an accuracy on the test data of 59.8 %, cross-validation accuracy 54.7 % and cross-validation standard deviation 3.15 %. The submissions gets a score of about 1.15.

There exists a clear performance gain when using FV-encoded features and thus higher order statistics. Based on this, the encoded scheme is further used in later models.

When plotting the learning curves (figure 5), it is clear that our optimized linear model performs well without any overfitting. However, due to the lack of sufficient data, the model can only perform so much on what it is given. Should the model have more training data at its disposal, the training score and CV-score will get closer and asymptotically reach the highest theoretical possible score for this model, which will be around 63.0 % accuracy.



(a) In function of accuracy.



(b) In function of log loss.

Figure 5: Learning curves of linear model BOVW.

When observing both normalized and unnormalized confusion matrices (figure 6) for both encoding schemes, it is clear that the Fisher Vector approach makes the model more confident in almost all classes, reassuring their value as encoding scheme. However, some very interesting observations can be made about the animal classes themselves. The model is very bad at predicting wolves, they get confused with mostly bobcats and persian cats, and in lesser form with lions and german shepherds. The most apparent visual interpretation for these errors are the pointy, upright ears that these animals share. Since wolves are the most underrepresented class in the train data, most of their image patches will cluster around this ’ear’ feature, causing the model to be confused a lot. Another interesting case is the confusion between chihuahua’s and siamese cats. Since these are pet animals, it is probable to assume that background feature play an important role in the confusion.

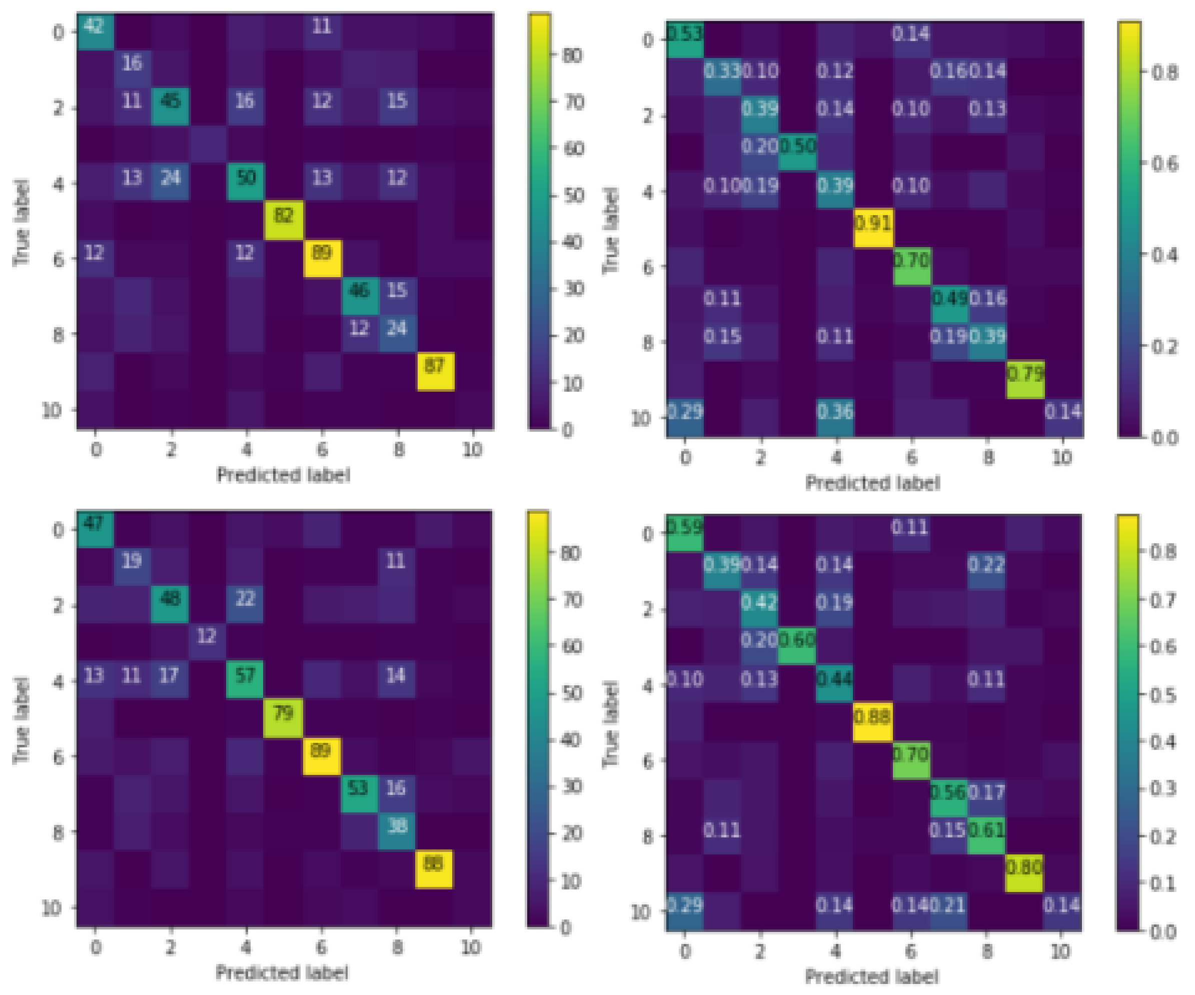


Figure 6: Confusion matrices of linear models with BOVW on top and with FVs bellow.

5.2 Non-linear Models

Gaussian Process Classifier

The first non-linear model that was used is a Gaussian Process Classifier (GPC). Gaussian Processes (GPs) are supervised learning methods commonly used for Regression and Classification problems, in which the latter returns the test predictions as class probabilities.

The GPC uses the Laplace approximation for approximating the non-Gaussian posterior by a Gaussian. One of the advantages is that the prior's covariance is set through a kernel object, having a wide variety of kernel functions available and optimizing its hyperparameters by maximizing the Log-Marginal-Likelihood (LML) makes it suitable for different types of problems.

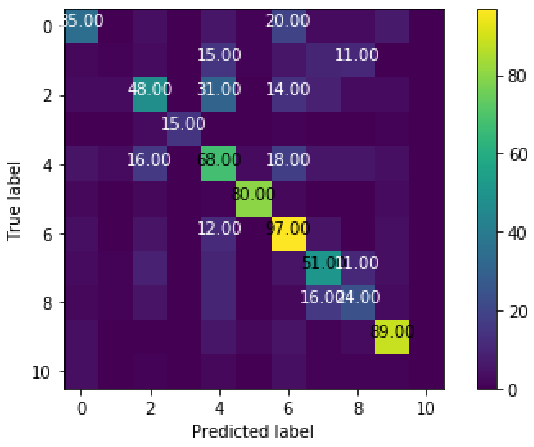
The GPC is able to perform one-versus-rest or one-versus-one training and prediction for multiclass problems, although one-versus-one only supports plain predictions not probability estimates, therefore we will use one-versus-rest.

The most influent hyperparameter is the kernel object so the next paragraph is dedicated to it. Other parameters have been selected to improve the speed and performance of the model such as: n_restarts_optimizer that will help the optimizer to not get stuck in local minima and max_iter_predict which is the number of iterations in Newton's method for approximating the posterior during predict. These values have been set to n_restarts_optimizer=3 and max_iter_predict=500 looking into a balance of computation time and good results.

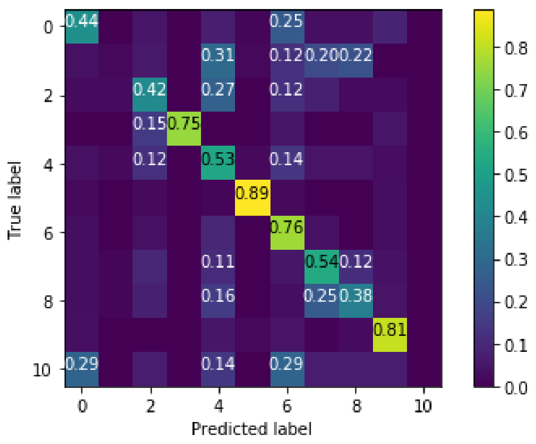
The kernel object is used to compute the GP covariance between two datapoints and thus will determine the shape of prior and posterior of the GP. The kernel is parametrized by a vector θ that will be optimized in terms of the LML. It is also possible to use a composition of several kernels, and the one that gave us best result is the RadialBasisFunction combined with a Constant kernel. Other combinations have been tried with the Matern, Rational Quadratic and ExpSineSquared kernel functions.

The Gridsearch was performed first on a variety of kernel objects and some simple combinations, having the best score with kernel=10*RadialBasisFunction(). As a second step, the hyperparameters in the kernel object are optimized in a new Gridsearch. RBF() has two hyperparameters: the length scale that allows to chose between an isotropic or anisotropic kernel, and its boundaries. The range used in the gridsearch is length_scale:{0.1, 0.5, 1, 50, 100}. The final value for this parameter coincide with its default value length_scale=1.0. Though it's important to mention that the kernel's hyperparameters are automatically optimized by the model itself, so the gridsearch acts more as a way to speed up convergence to the optimal values.

The confusion matrix for the models are plotted in figures 7 and 8, where we observe that the worst predictions are for wolfs and chihuahuas, most probably because the lack of training images for these classes. The persian and siamese cats are often confused due to their similarities.



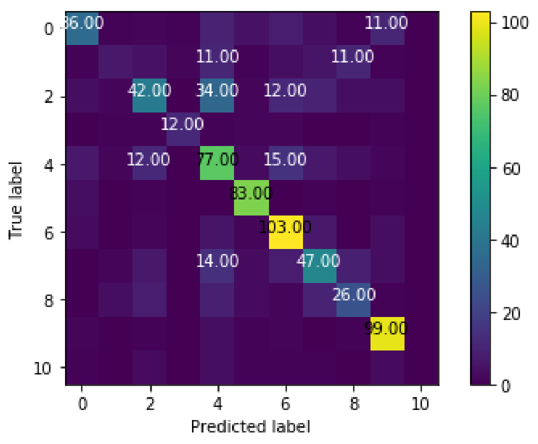
(a) BOVW features.



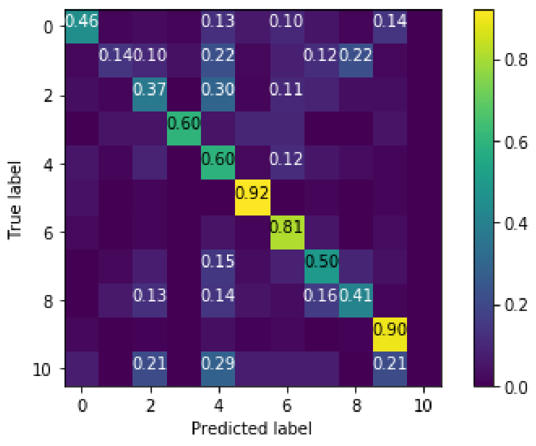
(b) BOVW features normalized.

Figure 7: Confusion matrix for GPC BOVW model.

The overall performance was slightly better in the BOVW model having a submission score of 1.19 compared to the FV model with 1.22. As we can observe from the learning curves in figure 9 for both models overfitting is kept at bay, while we see that the model could perform better in case of having a bigger training set of images. From



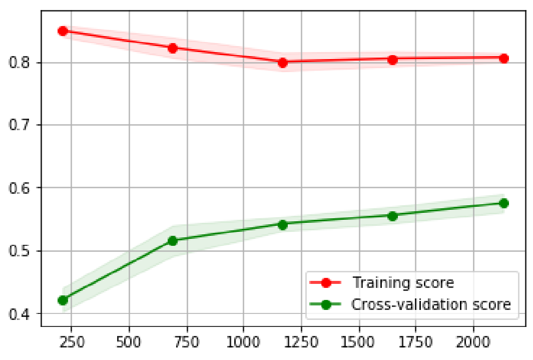
(a) FV features.



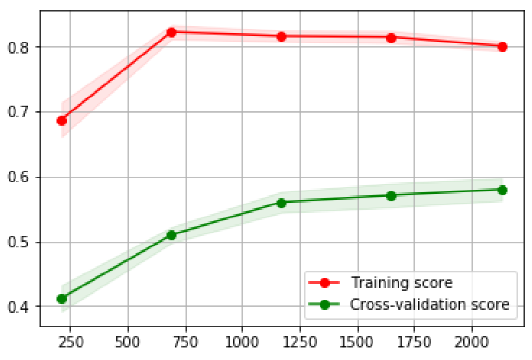
(b) FV features normalized.

Figure 8: Confusion matrix for GPC FV models.

the FV learning curve we see that the training score increases with the length of the training set most probably because its harder for the model to overfit to a bigger training set, though it could also mean there's too much regularization. On the other hand we observe that the BOVW tends to overfit slightly.



(a) GPC BOVW model.



(b) GPC FV model.

Figure 9: Learning curves for GPC models.

Non-Linear SVM

A second Non-linear model that we tried out is a non-linear Support Vector Machine (SVM), since this is a type of supervised learning that can be used for classification and that has returned decent image classification results in the past [3].

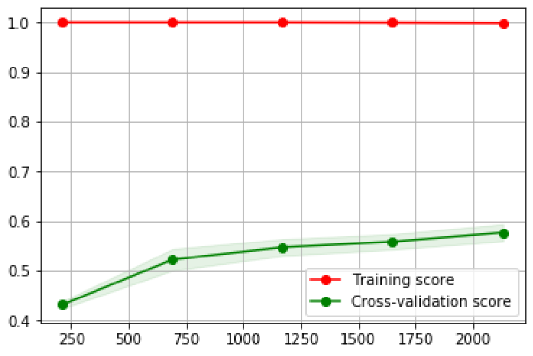
Skicit-learn provides two non-linear classes that serve this purpose: SVC and NuSVC. These are similar methods, but NuSVC uses a parameter ν to control the number of support vectors, while SVC uses a regularization parameter C that provides a trade-off between correctly classifying training examples and maximizating of the decision function's margin. Since the SVC variant seemed easier to interpret for optimization than the NuSVC one, SVC is the one we went with.

If the C parameter value is high, smaller margins will be accepted to make sure that most of the training points are classified correctly. For low values of C, there will be a larger margin (resulting in a simpler decision function) that might miss-classify more training points.

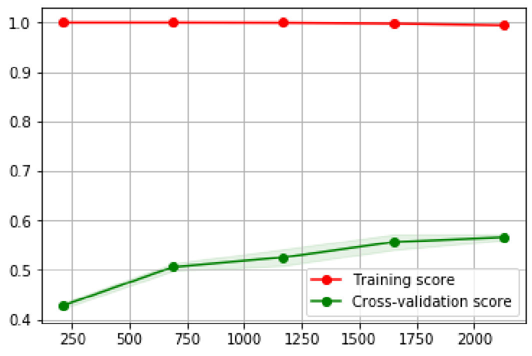
Another important parameter of the rbf kernel is 'gamma' (γ). This parameter defines how far the influence of a single training example reaches. If γ is low, this influence will reach far, and if γ is high, it's reach is close.

The SVC model allows compensation for the class-imbalance in our data by letting the parameter $class\_weight$ take on the value 'balanced'.

The SVC model provides 3 common non-linear kernels: 'poly', 'rbf' and 'sigmoid'. The 'rbf' (Radial Basis Function) kernel is the one that is used in most cases. When optimizing the model parameters using GridSearchCV, it was also the one that gave us the best results. Gridsearch (optimizing on log loss) was performed for C values in [1, 10, 100, 1000] and gamma values in [0.1, 1, 10, 100, 1000]. The returned values for both FV and BOVW features were: $C = 10.0$, $gamma = 1.0$. The used PCA threshold for dimensionality reduction was 0.6. Both the predictions for FV and BOVW features had pretty good log loss scores, and did well on Kaggle as well. When looking at the generated learning curves in figure 10 and train accuracies in table 1, however, it became apparent that using these values caused severe overfitting on the train data.



(a) FV features.



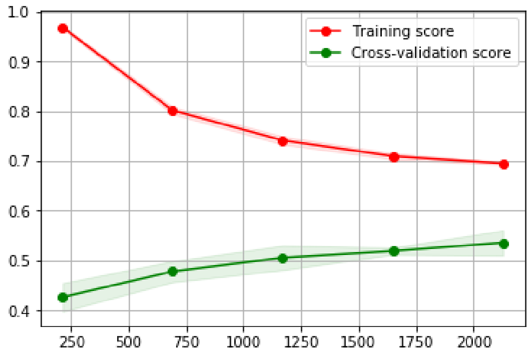
(b) BOVW features.

Figure 10: Overfitting learning curves for non-linear SVM.

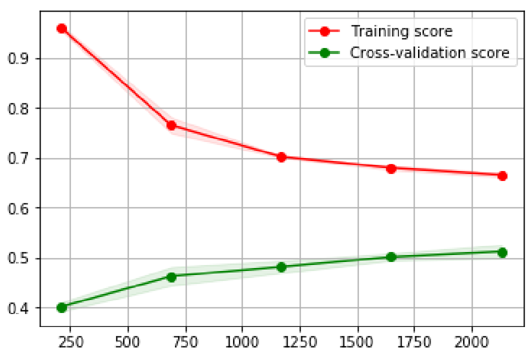
Overfitted model	Loss	Train	Test
Non-linear SVM BOVW	1.29	99.1 %	53.5 %
Non-linear SVM FV	1.19	100.0 %	58.2 %

Table 1: The performance values for the overfitting models.

Several things, like lowering the PCA threshold and further tuning of the hyperparameters were tried to avoid this overfitting. Since the main cause for overfitting was found to be that the value of gamma was too high. The influence of each training example did not reach far enough, causing the decision boundaries to be too "tight" around the data. Lowering this gamma to a value of 0.1 instead of 1.0, combined with decreasing the number of PCA components by lowering the threshold to 0.55 for BOVW and 0.50 for FV features, largely made up for the overfitting. The resulting learning curves are shown in figure 11. The curves are much closer together, so a lot of the previous overfitting was avoided here.



(a) FV features.



(b) BOVW features.

Figure 11: Improved learning curves for non-linear SVM.

The performance values for the manually improved hyperparameters are shown in table 2. We can see that indeed, the train and test accuracies are much closer together than before, while still maintaining decent log loss scores. The final results are a little (but not that significantly) better for the model trained on the Fisher Vector features than they are for the BOVW features, so the FV features are what was used in the handed-in code.

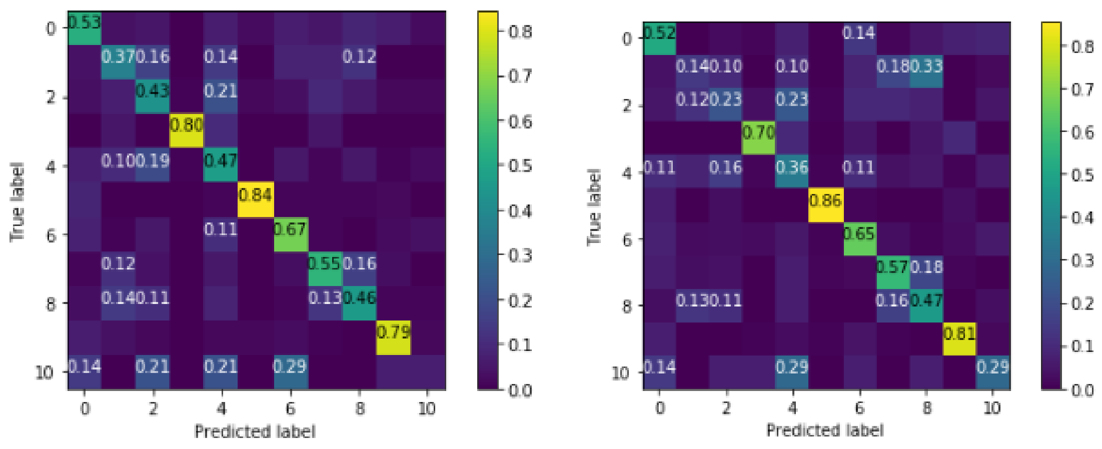
Model	Loss	Train	Test
Non-linear SVM BOVW	1.27	65.6 %	54.1 %
Non-linear SVM FV	1.26	68.2 %	55.7 %

Table 2: The performance values for the improved models.

For the final SVM models, the confusion matrices are plotted in figure 12. These tell us that the models struggle most to recognize wolves (10) and chihuahuas (1) and also that they tend to confuse the Persian (7) and Siamese (8) cats sometimes. As for wolves, this can be explained by the fact that there are not that many train images available. For the chihuahuas there were probably too many different subspecies that were mixed up with other animals. For another underrepresented class, the dalmatians (3), the models get decent results, which can probably be explained by the animals' very discernable spot pattern. The models are best at recognizing big cats like leopards (5), tigers (9) and lions (6), which probably has something to do with their fur patterns and the large amount of train images available for those classes.

5.3 Ensemble Techniques

As a short intermezzo, it is worth stating that ensemble methods were tried in order to achieve better performance. Out of the off-the-shelf models available through the sklearn library, RandomForestClassifier seemed the most promising. Also an external library (XGBoost) was used for evaluating their XGBClassifier, as it performed even



(a) FV features.

(b) BOVW features.

Figure 12: Normalized confusion matrices for non-linear SVM.

better. However, after optimizing the hyperparameters to *n_estimators=200*, *subsample=0.45* and *booster=dart*, the model only got a logloss of 1.37 with severe overfitting (leaderboard score of 1.33). The ensemble techniques did not seem promising and other options were prioritized, so the investigation for these kind of models was quickly stopped. For model evaluation of the XGBClassifier we would like to refer to the corresponding notebook, since they are not included in the report for conciseness and because other models were put as more important.

5.4 Convolutional Neural Networks

The convolutional neural networks (CNNs) below are created with Keras and a TensorFlow backend. They are trained on a NVIDIA GeForce 940M GPU and evalutated with TensorBoard.

Data augmentations that are used, are:

- Randomly rotating over maximum 20 degrees.
- Randomly shifting the width over maximum $20\,\%$ of the image width.
- Randomly shifting the height over maximum $20\,\%$ of the image height.
- Randomly shearing over maximum 20 degrees.
- Randomly zooming in over maximum $20\,\%$.
- Randomly flipping over the vertical axis.

Newly introduced parts of the images are then filled by looking at the nearest value.

$20\,\%$ of the train set is split of and used for validation. All RGB values of the train, validation and test set are also scaled down to values between 0 and 1 by the prepocessing function.

For the CNN, labels are encoded as one-hot vectors ([1,0,0,...]: bobcat, [0,1,0,...]: chihuahua, ..., [0,...,0,1]: wolf) instead of integer values.

The self-created CNN has the following structure. It is a sequencial model that first alternates a 3 times between 2 Conv2D and 1 MaxPooling2D layers. The output is then flattended with a Flatten layer and sent through a Dense layer. All these layers have a ReLu activation function. To get the final outputs, one more Dense layer with 11 outputs and a softmax activation function is needed.

The CNN is now first trained with the RMSprop optimizer during 20 epochs and batch size 16. Then the weights of the best model are reloaded and the model is trained 20 more epochs with the SGD optimizer. The learning rate is hereby set to 0.0001 and the momentum to 0.9. The batch size is again 16.

Now there is made use of a pretrained neural network. On top of a base model, a GlobalAveragePooling2D layer, a Dense layer with 1024 nodes, a Dropout layer with 10 % dropout, a Dense layer with 128 nodes and a Dense layer with 11 outputs and the softmax activation function are added.

For the base model, different options were tried out. VGG16 and VGG19 did not score that well. InceptionV3 scored way better, but was still outperformed by Xception. The model with Xception as base, was trained for 25 epochs with the RMSprop optimizer and fine tuned during 12 epochs with the same SGD optimizer as in the selfmade model. The batch size was kept at 16 images.

The results (log loss, train accuracy and test accuracy) in function of the amount of epochs for both models can be found in figure 13.

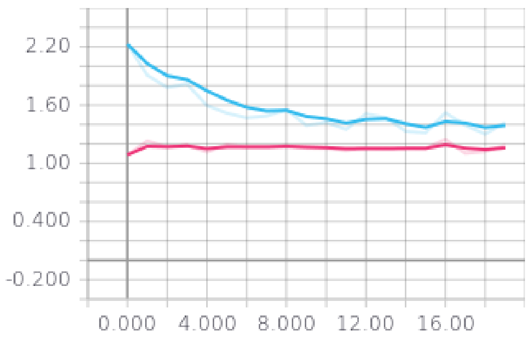
6 Conclusions

An overview of the scores of all classifiers can be found in table 3. CVA stands for cross-validation accuracy and CVS is short for cross-validation standard deviation. There can be concluded that, with less programming effort and approximately the same training time, Convolutional Neural Networks outperform both, linear and non-linear classifiers by far.

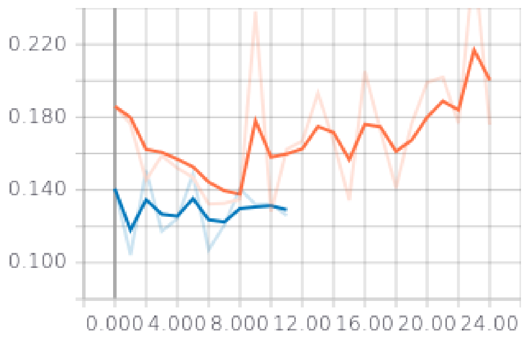
Model	Subm.	Loss	Train	Test	CVA	CVS
Linear Model BOVW	1.22	1.26	75.1 %	55.5 %	53.7 %	2.10 %
Linear Model FV	1.15	1.20	74.3 %	59.8 %	54.7 %	3.15 %
Non-linear GPC BOVW	1.19	1.30	78.9 %	58.6 %	55.3 %	2.65 %
Non-linear GPC FV	1.22	1.30	81.4 %	59.1 %	57.5 %	2.32 %
Non-linear SVM BOVW	NA	1.27	66.8 %	52.9 %	52.4 %	1.56 %
Non-linear SVM FV	NA	1.20	77.2 %	58.0 %	56.0 %	1.40 %
Ensemble XGBoost FV	1.33	1.37	98.5 %	53.7 %	52.5 %	0.94 %
CNN	1.02	1.09	64.5 %	62.9 %	NA	NA
CNN pretrained	0.07	0.10	97.5 %	96.3 %	NA	NA

Table 3: An overview of all models and their performance.

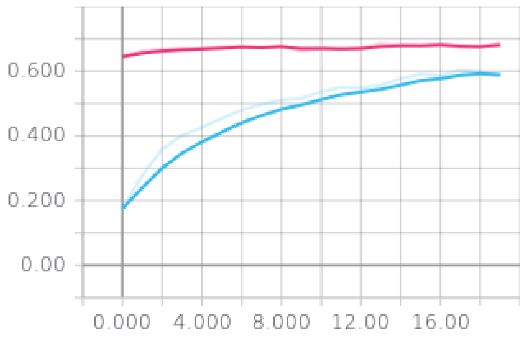
The biggest remaining problem with the linear models, is predicting wolves. More will need to be done against class imbalance. Another problem is that the models confuse



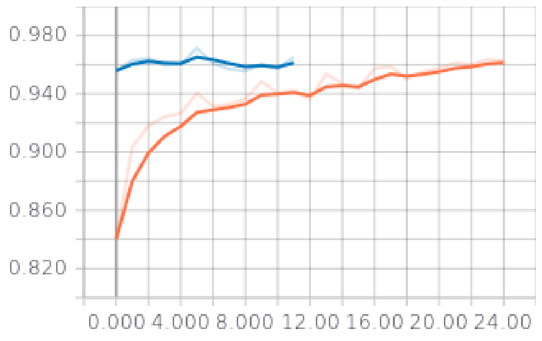
(a) CNN loss.



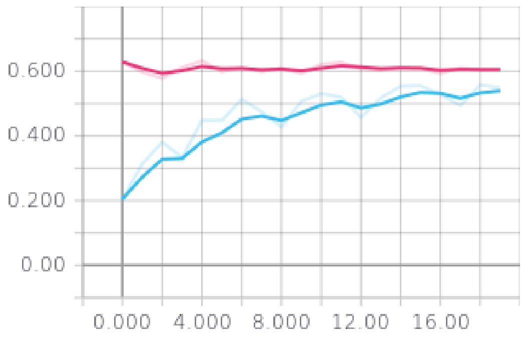
(b) Pretrained CNN loss.



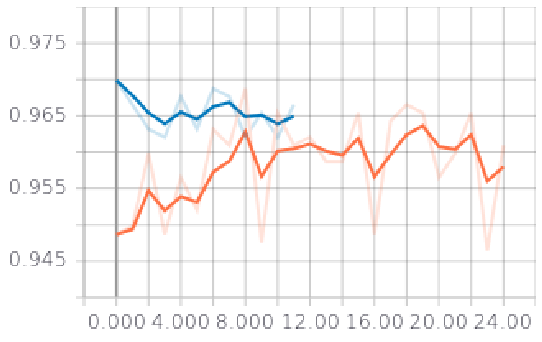
(c) CNN train accuracy.



(d) Pretrained CNN train accuracy.



(e) CNN test accuracy.



(f) Pretrained CNN test accuracy.

Figure 13: Results for CNNs.

collies and German shepherd way to often. A large improvement will be necessary to distinguish between them.

A way of improving current classifiers, is to combine data augmentation with multiple feature extractors and optimize more towards the codebook size and the amount of PCA features that are kept.

References

[1] S. L. J. Zhang, M. Marsza lek and C. Schmid, "Local features and kernels for classification of texture and object categories: A comprehensive study," *International Journal of Computer Vision*, vol. 73, no. 2, pp. 213–238, 2006.

[2] M. Seeland, M. Rzanny, N. Alaqraa, J. Wäldchen, and P. Mäder, "Plant species classification using flower images—a comparative study of local feature representations," *PloS one*, vol. 12, no. 2, p. e0170629, 2017.

[3] D. Masri, Z. Aung, and W. L. Woon, "Image classification using appearance based features," in *2015 11th International Conference on Innovations in Information Technology (IIT)*, pp. 128–133, Nov 2015.