Machine Learning challenge: final report

Stefano Claes

January 17, 2021

Contents

Abstract

Following document is a report on the programming pipeline that has been used to predict several animal species. We will discuss the pipeline from beginning to the end where we do the last classification of data. Because of performance issues we will only elaborate on findings of the logistic regression classifier. Indeed Support Vectors machines where an excellent solution for the problem we were facing.[1]

[1] In fact if you search GitHub for solutions to similar tasks you'd find that almost all people used SIFT [1] in combination with a support vector machine. Furthermore the same tendency continued if you'd search more academic places like `https://www.researchgate.net`.

1 Data exploration

Before everything else we would look to the data and see what we are dealing with, we start here by looking to the quality of the pictures.

1.1 Resolution of pictures

(a) Test-data image resolutions

(b) Train-data image resolutions

Figure 1: Image Resolutions

When looking at the resolutions we see the most occurring resolution is 720 by 1080 followed by 225 by 300. Depending on the descriptor we should resize the pictures to somewhere in between. For example when using the DAISY descriptor we also use a fixed amount of key points using the Shi–Tomasi corner detection algorithm.[2] In case we do not alter either the amount of key points for each resolution or set a resolution standard we get very bad detection results.

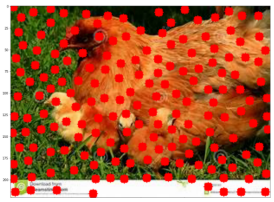


Figure 2: Default resolution with 500 Shi-Tomasi keypoints and min distance 15

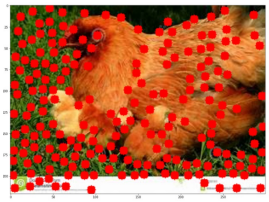


Figure 3: Default resolution with 250 Shi-Tomasi keypoints and min distance 11

Figure 4: Adjusted resolution and increase to 500 keypoints.

As seen on figure 3 and 4 the key points aren't just scattered all over the picture, this should increase the accuracy of our model since we are catching less irrelevant features that don't help us with classification.

1.2 Animal distribution

When having to learn a classification model a big problem is when having an imbalanced set of test-data, the majority class chicken in our case will be an animal that will be very easy to detect other animals like the golden retriever or the jaguar will be more difficult due to the fact we don't have enough examples. To counterfeit this multiple solutions are possible.

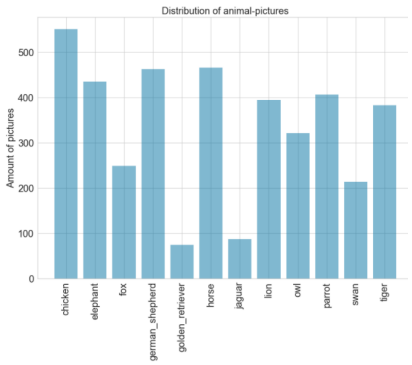


Figure 5: Distribution of animal-pictures

1.2.1 Solution 1) Adding the weight parameter to the classifier

When using a classifier you can opt to set the class weight to "balanced", this will use the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as n_samples / (n_classes * np.bincount(y))[3]. So during classification the classifier will make more changes to the model when classifying a sample of the minority class.

1.2.2 Solution 2) Adding artificial samples with SMOTE

Another solution would be to add artificial sample using oversample techniques like SMOTE.[2] Below we see a more visual example we have from `https://imbalanced-learn.org/`[4].

(a) Original data

(b) Data after using SMOTE

Figure 6: Using SMOTE to handle data imbalance

As we can see we would increase the data but there is no guarantee that we won't make data that makes no sense since we have an unsupervised problem.

1.2.3 Solution 3) Adding pictures using external sources

A third option is just to add samples from imagenet[5], this way we can download additional animals of each type. The result would be that more outliers would be found and a further increase of codebook would be possible.[3] From the 3 solutions I think this is the best one, I didn't try it due to the pkl files becoming very big and a PC that wasn't fit for the job. The reason why I think this is the best one is because this is the only technique which would fully exploit in-class variants. There might be pictures of the minority class taken from an unseen angle which results that the classifier won't know what to do with a test-sample.

1.3 Feature correlation

The question could be raised whether we could do some feature engineering, to quantify the need we show following correlation matrix.

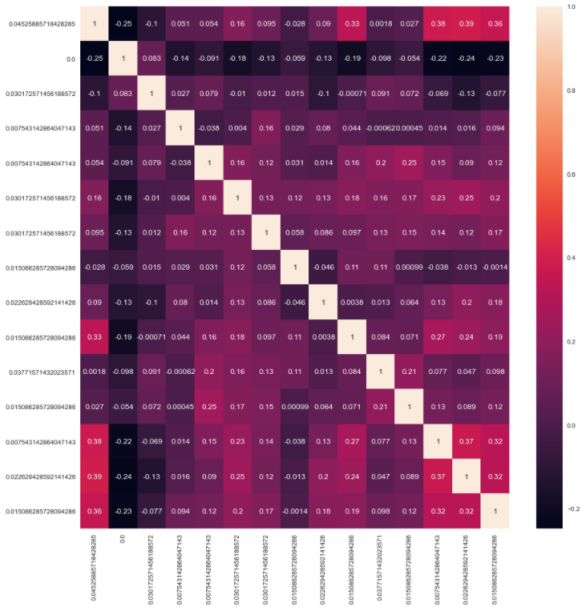


Figure 7: 15 most correlated features

[2] Synthetic Minority Oversampling Technique
[3] More on this later since this was a data-leak we overlooked in the code.

Since we didn't find much correlation we opted out to work further on feature engineering. Note that this correlation matrix is made after the binning[4] process of createcodebook.

2 Visual bag of words

A lot has been tried to enhance the visual bag of words, some things worked out some things made things worse. Since this is the basis of the classification pipeline improving here greatly increases the accuracy of the final model.

2.1 Feature matching

During this experiment we wanted to see if there was some correlation between the type of animal and the ratio of key points which were matched with the picture of the next animal in of its type. For this we used SURF[6]to match the corresponding key points the Brute-Force Matcher of OpenCV was used.[7]

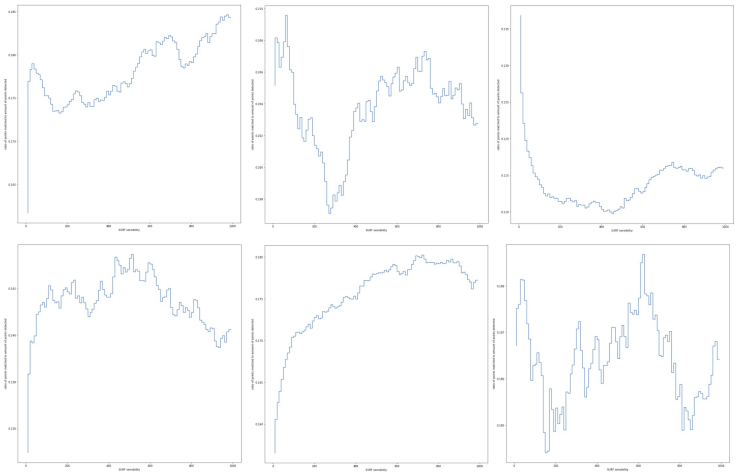


Figure 8: Ratio of matched points versus surf descriptor hessianThreshold

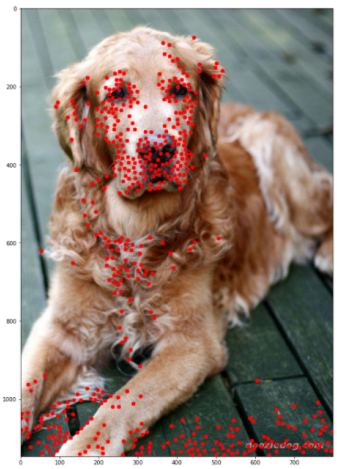
Only the results from the first 6 animals are included since it was not taken in account in further stages of the pipeline, the general idea was to put difference thresholds for different resolutions to extract the maximum amount of features. The code to generate this plot can be found in the appendix A.1 .

2.2 Daisy descriptor

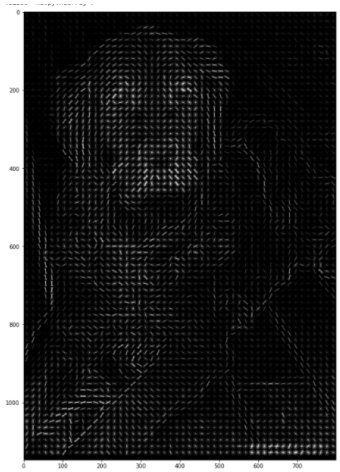
The first descriptor that was tested was the DAISY descriptor, improving this descriptor was possible by resizing all images to a standard of 1000 by 1000 and cropping the image to 800 by 800 so less background noise was captured. This enabled us to decrease our score to about 1.24. An improvement on this, which wasn't tested to great extent was improving this by adding Global features to this local descriptor[5].[8]

[4] This can be seen also as feature engineering.

[5] We did this but decreased all images to a size 256 by 256, it gave a better score with hog than without but didn't improve on the 1.24. Actually it should have been tested on the default set of pictures instead of scale down.



(a) Shi-Tomasi Corner Detection on an image



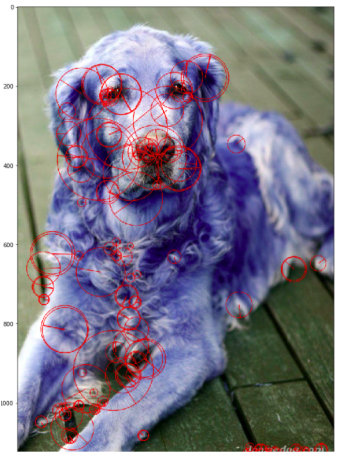
(b) Histogram of Oriented Gradients on an image

Figure 9: A local and Global descriptor on the same image

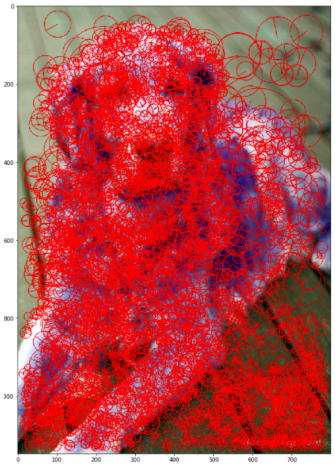
To use both information you need to adjust the encode image code, so it also takes the histogram of Oriented Gradients into account. A full explanation can be found here `https://github.com/flytxtds/scene-recognition/blob/master/scene_recognition_local_global_approach.ipynb`.

2.3 SURF descriptor

A second approach was the SURF descriptor which in fact was way more performing than the DAISY descriptor. Since the SURF descriptor finds key-points according to internal workings and the Hessian threshold we don't have to re-scale nor adjust the threshold for different types of resolutions.[6] The final pipeline used a Hessian threshold of 100 which catches on average several thousands of key points per picture. To give an example we will show the same dog one more time.



(a) Hessian treshold = 5000
amount of keypoints = 136



(b) Hessian treshold = 100
amount of keypoints = 4989

Figure 10: A comparision on using different values for Hessian threshold

When setting the threshold the preference should be given to the lower value because it captures more patches of the dog, yes it catches more from the background but a lot more of the dog itself is caught and that's beneficial for creating a vbow. The higher value gives one mayor benefit, it makes it possible to see if the descriptor is catching the right patches of the image.

2.4 Codebook creation

The standard codebook made about 500 features which determined the amount of visual words we would use to describe each animal. In general, we would want to keep this a small value due to Occam's razor or the minimum description length principle[7].[9] testing with log loss score gave us a rather large value of 2500 and this is what we kept using during the entire pipeline. In fact this is one of the **mayor mistakes**, when searching for the optimal codebooksize you should split the test and train data beforehand. In this way the validation set is a set which hasn't been clustered on[8]. Since this is an unsupervised problem it is unclear on what parameters the k

[6] As seen in the plot's where we varied the hessian threshold

[7] To give an example, we want a dog to be classified as a dog due to features around his tongue or his ears, not due to the fact that most dog pictures are taken near grass or in snow.

[8] An avoidance of a memory leak

means clustering is making decisions. Enlarging the codebooksize will make the elements on which we decide which animal it is more and more specific it is an actual visual book since it consists of little elements which the descriptors have found.

2.4.1 Preprocessing

2.4.1.1 Standardization

Before k-means we should standardize since the variables from the descriptors have unrelated values. This is done by applying the function below for each attribute of each element in the test data.

$$Z = \frac{x - \mu}{\sigma}$$

2.4.1.2 Normalization

During testing multiple preprocessors were tested, of all this scaling techniques Normalization gave the best results.

$$X_{new} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

The reason this gave better results or one of the probable reasons is because here we not only eliminate the fact that some values grow faster than other values, we also eliminate outliers that would lead to bad clusters.

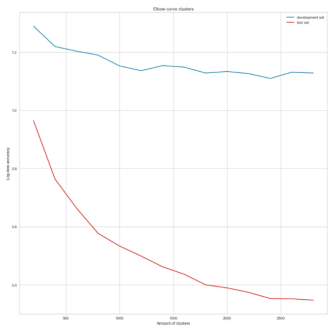
3 Classifiction

Many classifiers where tried, so we will only elaborate on the classifiers which were in the final voting classifier. In fact, we will not go in to a too deep detail on the XGboost classifier since we just followed an online reference. When performing the prediction a particular scheme was followed which you can find in the Appendix A.2. The code you will find there uses stratified k fold as test train splitting technique. This ensures that the same percentage of each class is used when fitting the model, a major difference with a normal train test split that won't take this into account and will just split at random.

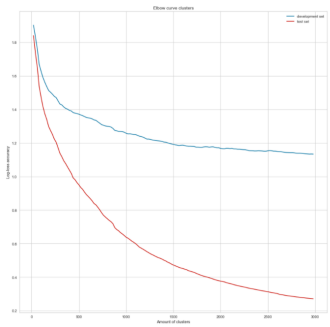
3.1 Logistic regression

Logistic regression was the go-to classifier for fast computation, this enabled us to test new pkl files, new preprocessing strategies, look for optimal codebooksize , viewing the cross validation curve.

3.1.1 Codebook finetune



(a) Elbow curve on multiclass log loss



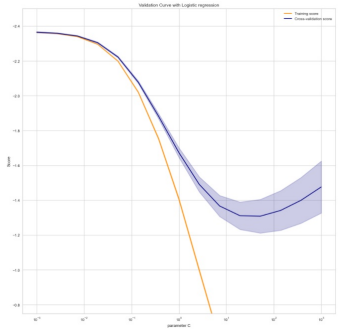
(b) Selecting the k best values using anova as feature selector

Figure 11: Two cross validation curves on scoring log loss ranging over the number of features

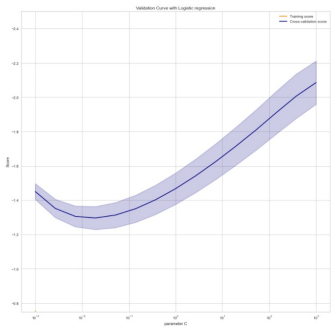
Here we wanted to find out what the optimal codebooksize for our problem is, now bear in mind no split was performed and this is why in figure a, it is possible to see improvement until 2400 visual words.[9] On figure b we wanted to know if we could slim down our dimension, so we had fewer variables performing better predictions, unfortunately there was no such trend and taking all the visual words gave us the best predictions. Code can be found in apendix A.4.

[9] The graph starts form 200 words

3.1.2 Cross validation curves



(a) Cross validation curve without preprocessing



(b) Cross validation curve using yeo-johnson power transform

Figure 12: Two cross validation curves on scoring log loss ranging over the parameter C

Both curves touch the ranges of 1.2 and 1.4. Furthermore, the graph's change in 2 significant ways. After the Yeo Johnson transformation the regularization parameter C is much smaller at its optimal value and the variance is now constant. Both are normal consequences since the data has to be less regularized for overfitting on big variances. Because of this we can keep the C parameter too a much lower value.

3.1.3 Gridsearch

C = np.logspace(-3, 3, 15)[10]

3.1.4 Parameters

C = 20

3.1.5 Projected multiclass loss

1.08 +/- 0.014
The reason why we lost score on kaggle where presumably due to following reasons which apply to every forthcoming classifier.
1) No split in data before clustering
2) No weighted parameter nor efforts to make data more balanced

3.2 Support Vector Machines

Support vector machines was somewhat a standard to use when doing image classification using a bag of features. Almost all GitHub repo's and acedemic papers used this as the final step of the pipeline. The results we got from it where not outstanding presumably because we didn't hypertune it that much with gridsearch. In fact we did a gridsearch on a very sparse manner and didn't look for some local minimum.

3.2.1 Gridsearch

C = [50, 100, 1000]
gamma = [1e-2, 1e-3, 1e-4, 1e-5]

3.2.2 Parameters

C = 1000
gamma = 0.01

3.2.3 Projected multiclass loss

1.074 +/- 0.02

3.3 XGboost

XGBoost is an optimized distributed gradient boosting library.[10] Because this library is optimized to not overfit it is a usable gradient boosting classifier. Even more since it works on a very different way than the svm classifier it was interesting to make an ensemble to improve the score even more. The finetuning of this model was realized by following the next link: `https://www.analiticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/`

[10] More parameters can be tuned using logistic regression but in earlier findings the difference was rather small

3.3.1 Gridsearch

Due the fact you have to finetune too many parameters here, either u incrementally fine-tune as opposed by the link or use something like randomized grid search.

3.3.2 Parameters

objective = multi:softprob , eval metric = mlogloss , colsample bytree = 0.8 , scale pos weight = 1 , learning rate = 0.01 , reg alpha : 1^{-4} , max depth = 6 , min child weight = 2 , n estimators = 2000[11] , subsample = 0.8

3.3.3 Projected multiclass loss

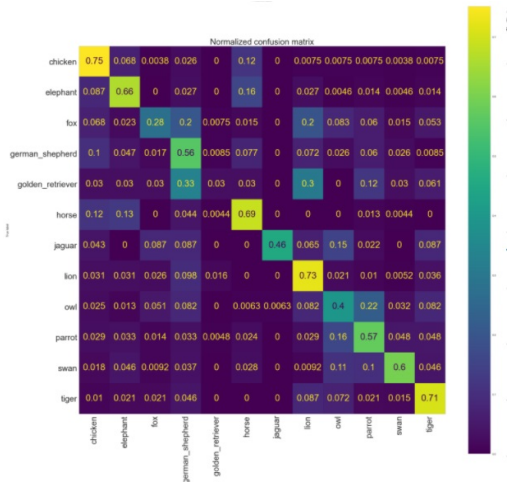
Unknown for SURF.

3.4 Voting Classifier (Ensemble classifier)

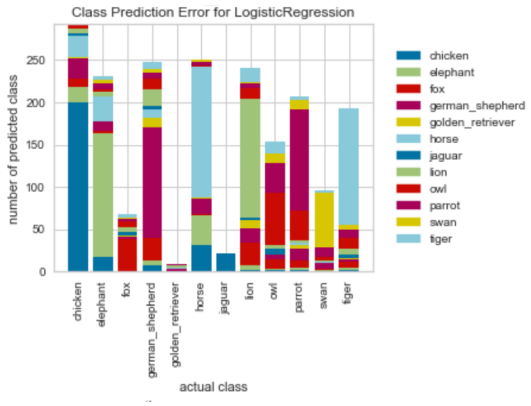
To finalize and take all the best things from every classifier an ensemble classifier was used. This classifier can be found in appendix A.3. We used soft prob instead of hard voting because this would increase our log loss metric the most. Finding the optimal weights takes a very long time and this loop should only be performed when all classifiers are finetuned properly. It was also this model which was used as a final score.[12]

4 Results

In this section the results that a logistic regression classifier made without pipeline and the C parameter set to 20. For the experiments we did a 50 percent test/train split.



(a) A normalized confusion matrix

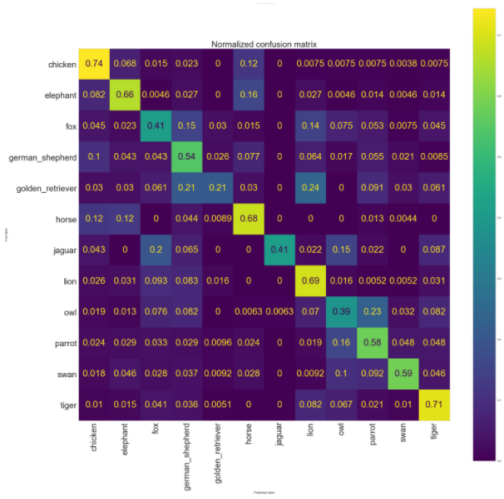


(b) A class prediction error visualization.[11]

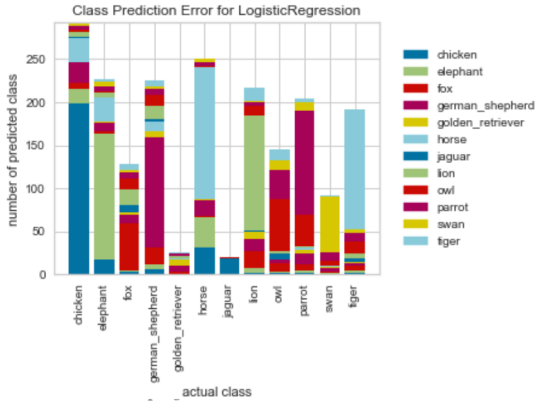
Figure 13: Logistic regression without class imbalance techniques

As predicted the classifier is unable to detect animals from the minority class, in fact 60 percent of the Golden retrievers get misplaced as either a German-shepherd or a lion. This is due to the fact that the animal patches aren't exploited to the full extend. The same goes for a fox but with lower values. To counterfeit this we will use SMOTE to create new samples only in the development set.[13] Also more can be seen by viewing the class error visualization, we see that the majority class chicken is represented in all animals, something that is most due to the fact it is the majority class not because it has the same looks like a tiger for example.

[11] For faster training you need to increase learning rate to decrease this value
[12] Due to lack of time the weights of when we were still using the DAISY descriptor were used
[13] As said before better solutions exist.



(a) A normalized confusion matrix



(b) A class prediction error visualization.

Figure 14: Logistic regression with class imbalance techniques

After performing class imbalance[14], we see fox and golden retriever get recognized 20 percent more of the time. This demonstrates that class imbalance is a mayor problem and action should be taken to counterfeit it. At last we look to the learning curve that is graded on a weighted f1 score.

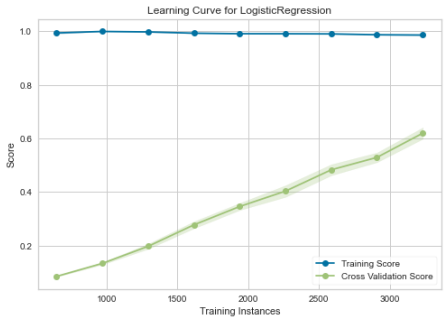


Figure 15: Learning curve based on weighted f1 score

The learning curve contains a data leak due to the fact that a train test split didn't happen before clustering, so we can ignore the perfect training score, the cross validation curve suggests that we aren't over-fitting and that more data would be beneficial to learn the concept even more.

5 Conclusions

During the time working on this project a lot has been learned. For example when and why do you normalize before k-means clustering, what is a data leak, how to handle imbalanced data. How to work with OpenCV , how to work with sklearn and many things more. A lot has been tried to improve the score, a lot failed and an equal amount didn't fail and progressed the score bit by bit. All by all this experience made me more experienced in sklearn and other libraries. If I were given this task again I would never start the same way I would say.

6 Moral Conclusions

I really liked the project as you can probably read, I enjoyed every moment of it except when my pc decided to stop for no reason. Now during the time I was working many people saw my place on the leader-board and knowing I'm kinda the class clown I got a lot of questions from people who were stuck and didn't really know what to do. Off-course I helped them bit by bit but for a next edition I would recommend having a more vibrant discussion page. For example a long time I was pushing the DAISY descriptor to the limit while by just switching to the SURF descriptor simple Logistic regression was able to beat my ensemble classifier[15]. I hope this isn't viewed from a negative angle, this task was as I said before very amusing and probably one of the key-factors to choose the AI option for my master's degree.[16]

[14] $count\_class\_golden\_retriever = 250$
$count\_class\_fox = 250$
$sm = SMOTE(sampling\_strategy = \{2 : count\_class\_fox, 4 : count\_class\_golden\_retriever\}, random\_state = 2)$

[15] Which isn't too complex if you compare it too kaggle winning solutions

[16] And yes I know, the master will not be a kaggle challenge. But preceding AI courses also peaked my interest.

A Python code

A.1 Feature matching code

```
total_kp = 0
best_ratio = 0
list_animal_match = []
ratio = 0

for label_string in label_strings:
    list_animal_match = []
    sensibility = []
    for i in range(10,1000,10):
        for j in range(1,10):
            current_image = cv2.imread(train_paths[label_string][j])
            current_image2 = cv2.imread(train_paths[label_string][j+1])
            current_image = cv2.cvtColor(current_image,cv2.COLOR_BGR2GRAY)
            current_image2 = cv2.cvtColor(current_image2,cv2.COLOR_BGR2GRAY)
            # create BFMatcher object
            bf = cv2.BFMatcher(cv2.NORM_L1, crossCheck=True)
            # Match descriptors.
            sift = cv2.xfeatures2d.SURF_create(i)
            kp1 , des1 = sift.detectAndCompute(current_image,None)
            kp2 , des2 = sift.detectAndCompute(current_image2,None)
            if (len(des1) or len(des2)) == 0:
                print("no keypoints where found")
            else:
                matches = bf.match(des1,des2)
                nominator = len(kp1)
                nominator = nominator + len(kp2)
                ratio = (len(matches) / nominator) + ratio
        ratio = ratio / 10
        list_animal_match.append(ratio)
        sensibility.append(i)
    %matplotlib inline
    plt.figure(figsize=(15,15))
    plt.xlabel("SURF sensibility")
    plt.ylabel("ratio of points matched to amount of points detected")
    plt.step(sensibility, list_animal_match)
    plt.show()
```

A.2 Prediction code format

```
param = {'C':np.logspace(-3, -1, 15)}
for train_index,test_index in kf.split(X,Y):
    print('\n{} of kfold {}'.format(i,kf.n_splits))
    xtr,xvl = X.loc[train_index],X.loc[test_index]
    ytr,yvl = Y.loc[train_index],Y.loc[test_index]
    model = Pipeline([('yeo', PowerTransformer(method='yeo-johnson')),
    ("log", LogisticRegression(C = 20 , max_iter= 2000))])
    model = GridSearchCV(model,param_grid, cv=3, scoring=LogLoss,iid=True)
    model.fit(xtr, ytr.values.ravel())
    print (model.best_params_)
    pred=model.predict_proba(xvl)
    print('logloss_score',log_loss(yvl,pred))
    pred_test = model.predict_proba(test_data)
    pred_test_full +=pred_test
    i+=1
```

A.3 Voting Classifier

A.3.1 Voting Classifier finetune

```
##clf 1 will become optimized svm
##clf 2 will become optimized logreg
##clf 3 will become optimized xboost
for w1 in range(1,5):
    for w2 in range(1,5):
        for w3 in range(1,5):
            X = train_data
            Y = train_labels
            X, X_DEV , Y , Y_DEV = train_test_split(X,Y,test_size = 0.5,random_state=42)
            #X = SelectKBest(chi2, k =  1920).fit_transform(X,Y)
```

```
#sm = SMOTE(random_state=2)
#X_DEV, Y_DEV = sm.fit_sample(X_DEV, Y_DEV)
X_DEV_arr = np.array(X_DEV)
Y_DEV_arr = np.array(Y_DEV)
X_arr = np.array(X)
Y_arr = np.array(Y)
clf1 = SVC(C=100 , gamma = 0.01 , probability = True)
clf2 = LogisticRegression(C= 8 , max_iter= 2000 )
clf3 = XGBClassifier(objective = "multi:softprob",
                        eval_metric = "mlogloss",
                        colsample_bytree = 0.8,
                        scale_pos_weight = 1,
                        learning_rate = 0.01,
                        reg_alpha = 1e-4,
                        max_depth = 6,
                        min_child_weight = 2,
                        n_estimators = 2048,
                        subsample = 0.8,
                        verbosity = 1)
eclf = VotingClassifier(estimators=[('svm', clf1),
('lr', clf2),
('xgb', clf3)],
voting='soft',
weights=[4,4,2])
eclf.fit(X_DEV_arr,Y_DEV_arr)
test_data_arr = np.array(test_data)
Forest_prediction = eclf.predict_proba(test_data_arr)
y_pred=eclf.predict(X_arr)
ada_forrest_train = eclf.predict_proba(X_arr)
acc_rf = log_loss(Y, ada_forrest_train)
print("-------------------------------------------------")
print("w1:" + str(4))
print("w1:" + str(4))
print("w1:" + str(2))
print("current log loss: " + str(acc_rf))
```

A.4 Select K best features

```
from sklearn.preprocessing import MaxAbsScaler
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2 , mutual_info_classif , f_classif
X = train_data
Y = train_labels
def plot_feature_selection_graph_anova(X, Y, minimum, maximum , step):
    best_score = 99999999
    idx = 0
    amount_of_zeroes = (maximum-minimum) // step
    DEV_errors = np.zeros(amount_of_zeroes)
    X_errors = np.zeros(amount_of_zeroes)
    for k_param in range (minimum,len(X[0]),step):
        X = train_data
        Y = train_labels
        X = SelectKBest(f_classif, k =  k_param).fit_transform(X,Y)
        X, X_DEV , Y , Y_DEV = train_test_split(X,Y,test_size = 0.5,random_state=42) #0.8 split gave us optimal results!
        logreg_best = LogisticRegression(C = 20 , penalty = 'l2' , max_iter= 2000 )
        logreg_best.fit(X,Y)
        logistic_regression_dev = logreg_best.predict_proba(X_DEV)
        acc_logreg_dev = log_loss(Y_DEV, logistic_regression_dev)
        logistic_regression_test = logreg_best.predict_proba(X)
        acc_logreg_test = log_loss(Y, logistic_regression_test)
        DEV_errors[idx] = acc_logreg_dev
        X_errors[idx]=acc_logreg_test
        print("---------------" + str(acc_logreg_dev))
        if acc_logreg_dev < best_score:
                best_score = acc_logreg_dev
                print("this is the featuresize: " + str(k_param))
                print("this is the best score: " + str(best_score))

        idx = idx + 1


    plt.figure(figsize=(15, 15))
    plt.plot(np.arange(minimum,maximum,step),DEV_errors,c='b',label='development set')
    plt.plot(np.arange(minimum,maximum,step),X_errors,c='r',label='test set')
    plt.xlabel("Amount of clusters")
    plt.ylabel("Log-loss accuracy")
    plt.title("Elbow curve clusters")
    plt.legend()
    plt.show()
    return best_score
plot_feature_selection_graph_anova(train_data, train_labels, 20,3000, 20)
```

References

[1] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, November 2004.

[2] Engin Tola, Vincent Lepetit, and Pascal Fua. Daisy: An efficient dense descriptor applied to wide baseline stereo. *IEEE transactions on pattern analysis and machine intelligence*, 32:815–30, 05 2010.

[3] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[4] Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, 18(17):1–5, 2017.

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[6] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer Vision and Image Understanding*, 110(3):346 – 359, 2008. Similarity Matching in Computer Vision and Multimedia.

[7] OpenCV. Open source computer vision library, 2015.

[8] Jobin Wilson and Muhammad Arif. Scene recognition by combining local and global image descriptors. *CoRR*, abs/1702.06850, 2017.

[9] Aditya Vailaya, HongJiang Zhang, Changjiang Yang, Feng-I Liu, and Anil Jain. Automatic image orientation detection. *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society*, 11:746–55, 02 2002.

[10] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM.

[11] Benjamin Bengfort, Rebecca Bilbro, Nathan Danielsen, Larry Gray, Kristen McIntyre, Prema Roman, Zijie Poh, et al. Yellowbrick, 2018.