Project-II by Group ATHENS

Giannakopoulos Athanasios
Kyritsis Georgios
EPFL
athanasios.giannakopoulos@epfl.ch
georgios.kyritsis@epfl.ch

Abstract

In this report, we summarize our findings for Project-II of the PCML course. The purpose of the project is to build an image classifier that predicts labels for unseen data. We experiment using neural network (NN), support vector machine (SVM) and random forests (RF) classifiers. We also investigate a few feature transformations and comment on the results.

1 Introduction

The goal of this project is to train classifiers that will be able to classify the object in a picture. To achieve this goal, we will practice on analyzing data and train several models.

Logistic regression, k-NN, NNs, RF and SVMs are among the most common classification methods. We focus only on NN, SVM and RF classifiers. In general, NNs perform well with images, so we expect a good performance for our classification task. Moreover, the stochastic training process of the NNs (train in batches, not in the whole data set at once) helps in performance increase. However, NNs are prone to overfitting. As far as SVMs are concerned, we expect good performance with linear kernel because of the big dimensionality of our data set. However, gaussian and polynomial kernels are expected to be very computationally expensive for our data set (due to its big dimensionality). Finally, RF are expected to be fast since multiple trees are trained simultaneously and averaged. The averaging prevents also from overfitting. However, we cannot be sure *a priori* for the performance of a method, since it often depends on the data set. Therefore, we test each one of the aforementioned classifiers.

Section 2 focuses on exploratory data analysis, presents the performance evaluation metric we use and introduces our baseline model. Sections 3 to 6 analyze the different classification models we use. Section 7 summarizes our results and presents the optimum values for the hyper-parameters for each model. Finally, we conclude in Section 8.

2 Exploratory data analysis

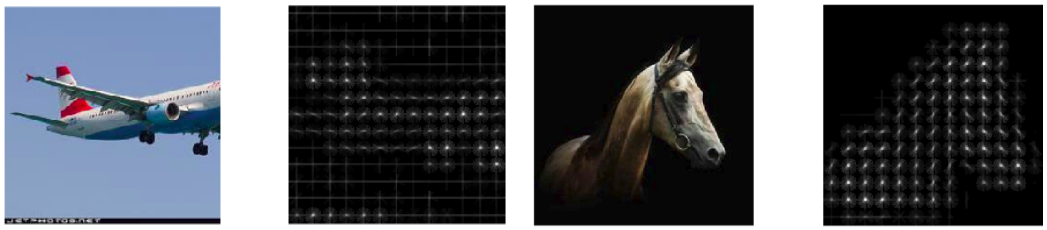
The training data set consists of grayscale and color images of size 231x231 pixels that contain either an airplane, a car, a horse, or none of the previous ones. Each picture $\mathbf{x_i}$ is associated with a label y_i. The depicted objects with their respective labels are shown in Table 1.

	Airplane	**Car**	**Horse**	**Other**
Label	1	2	3	4
% in data set	16.1	19.3	24.9	39.7

Table 1: Depicted objects and their respective label. The percentage of images in the data set depicting each object is also presented

For the provided images, we are given two sets of features[1]:

[1]The way these features are generated given an image is outside the scope of this report.



(a) Airplane HOG features (y=1) (b) Horse HOG features (y=3)

Figure 1: HOG features visualization

1. **Histogram of Oriented Gradients (HOG):** These features are often used in computer vision for purposes of object detection and try to describe the appearance and the shape of the depicted object using the distribution of intensity gradients and edge directions [1].

2. **OverFEAT ImageNet CNN Features:** These features are produced by training a Convolutional NN on a huge image data set. More information can be found in [2].

The total number of pictures is $N = 6000$ and the dimensionality D of the real-valued input matrix $\mathbf{X}$ varies according to the type of features we choose in order to train the classifier. Hence, it is $D_{HOG} = 5408$ and $D_{CNN} = 36865$ when HOG and CNN features are used respectively. Finally, we do not find any missing values, notice that HOG features are already normalized and that CNN features are sparse, i.e. many entries of $\mathbf{X}$ equal to zero.

Regarding the labels $\mathbf{y}$, we notice noise in the data, in the sense that numerous images are misclassified (e.g. image 458 depicts a horse but is classified as "other"[2]. The results we provide are upper bounded, in the sense that they can only be improved in case noise is not present.

Given is also a test set of 11453 images without any labels. Our goal is to classify these images as accurate as possible.

2.1 Data Visualization

We visualize the HOG features to get an understanding of what we believe the classifier learns from them. In Fig.1, we see how the derived gradients form the shape of the object in the picture. Color contrast seems to be of significant importance in order for the object contours to be clearly visible. For example, in Fig. 1(b), the horse is clearly distinguishable as it has a black background, whereas the airplane in Fig. 1(a) is less distinguishable in the HOG representation. We believe that the classifier tries to learn from common contours, edges and gradient directions, that appear in images belonging to the same class.

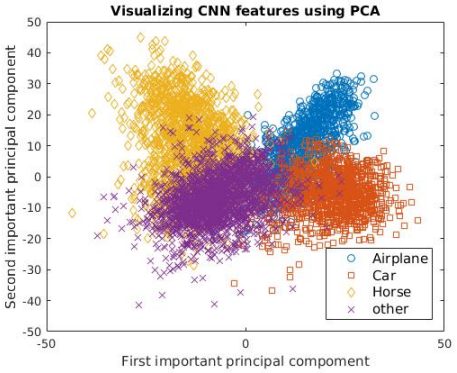
To visualize CNN features, we do PCA after normalizing $\mathbf{X}_{CNN}$ and keep the two most important principal components. The results are depicted in Fig. 2(a). We observe that the 4 different categories of objects form 4 different clusters. We believe that the classifier tries to find boundaries between these clusters when CNN features are used. Fig. 2(b) is representative for the value distribution of the CNN features. We see that the distribution is skewed and may contain outliers. We will try to tackle outlier in later sections.

2.2 Feature Engineering-Transformation and Dimensionality Reduction with PCA

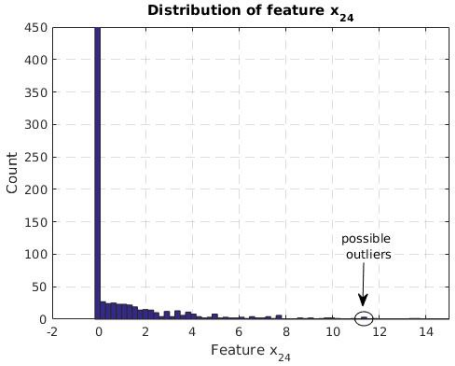
We normalize the input matrix $\mathbf{X}$ for all methods/classifiers that we are going to use. In addition, we look for near zero variance (NZV) in our features and remove those that display constant variance in the 99% of their data, since they do not carry information content with respect to the outcome. Moreover, we try centering and scaling of the features. However, this transformation presents the drawback of model interpretation loss, since variables loose their original meaning due to scale differentiation. We also merge the two set of features (HOG and CNN) together and test if we notice any performance improvement.

Finally, we apply PCA in order to decrease the dimensionality of $\mathbf{X}$. Hence, we aim at fighting the *curse of dimensionality* problem, especially in the case of CNN features, where $D \gg N$. Section 6 describes our methodology for determining the number of principal components d we keep. We train our classifier with both $\mathbf{X}$ and $\mathbf{X_{PCA}}$ and compare the respective performance.

[2]Due to time constraints, we do not re-classify all images of the data set.



(a) Visualizing CNN features using PCA (2 most important components)



(b) Distribution of CNN feature x_{24}. Skewed distribution with possible outliers. The y-axis is limited to 450 to ease visualization (the count is more than 5000)

Figure 2: CNN features visualization

All the aforementioned methods (apart from normalizing $\mathbf{X}$) will be applied only (due to lack of time) for the model that will give the best performance when the original $\mathbf{X}$ is used. Hence, we wish to see if we can achieve an even better performance. We apply these methods in Section 6 and present the results in Section 7.

2.3 Performance Evaluation

In classification problems, we can use different performance measures such as the accuracy ($loss_{01} = 1 - accuracy$) and the balanced accuracy ($BER = 1 - balanced\ accuracy$). In case of unbalanced data (see also Section 2.4 and Table 1), it is preferred to use the BER since it avoids inflated performance and allows for derivation of meaningful confidence intervals as stated and explained in [3]. Finally, we will test the performance of our models for both binary and multiclass classification, as explained in the project assignment.

2.4 Baseline Model

We will use a naive classification rule to obtain a baseline for our tested models. More concretely, we will always predict the most frequent class in our data set. For binary classification, the most frequent is the positive class, i.e. we will always classify a new image as positive with $loss_{01} = 0.397$ and $BER = 0.5$. For multiclass classification, the most frequent class is "other", i.e. we will always predict $y = 4$ with $loss_{01} = 0.603$ and $BER = 0.75$. Note that in both cases the classification does not depend on $\mathbf{X}$. For the rest of the analysis, we will use only the BER as a performance measure[3].

2.5 Splitting the Data Set

From Table 1, we see that the data set is unbalanced, i.e. all classes are not represented by the same number of images. In order to train our classifier, we build training sets that contain 80% of the images of each class and test sets that contain the remaining 20%, i.e. use stratified cross-validation to ensure each fold is a good representative of the whole data set. We test the performance of our method and it seems that the BER on the test set is better and with lower variance compared to random or balanced (same number of pictures per class) 80%/20% splits.

3 Image Classification with Neural Networks

Apart from the data set, given is a classification algorithm that uses a NN for image classification. This section intends to describe how this classifier works and to explain and estimate its hyperparameters. Final results are presented in Section 7.

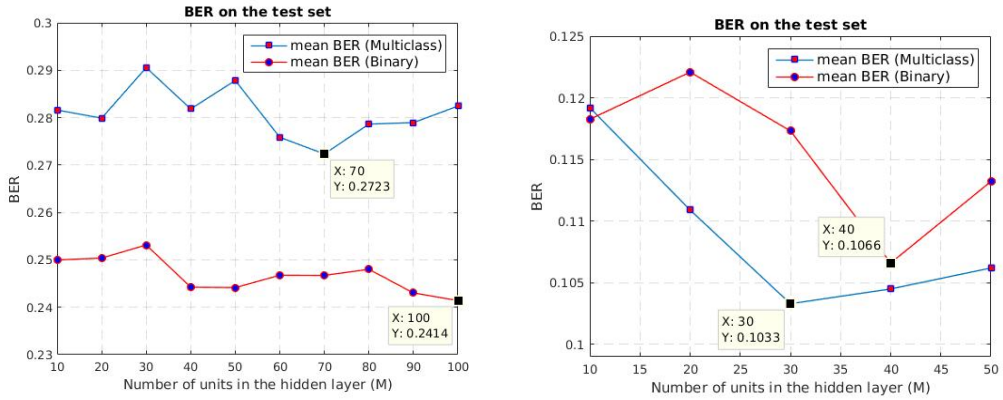
A NN consists of neurons (or units) arranged in layers that convert the input vector $\mathbf{x_i}$ into an output vector $\mathbf{y_i}$. The number of units in the input layer equals to the dimensionality D of the input matrix $\mathbf{X}$. Once again, D depends both on the use of HOG or CNN features and the use of PCA or not. The number of the output units equals to the number of the classes, i.e. 4. We denote with M the number of units in the hidden layer[4]. The more we increase M, the more our model overfits [4]. We

[3] BER is the performance metric that should be used as stated in the project assignment.

[4] Only one hidden layer used since more layers gave similar or worse performance.

Parameter	interval	step size	optimal value	meaning
learning rate	[0.2, 2.4]	0.2	2	controls the size of weight and bias changes in learning of the training algorithm
batch size	[20, 300]	10	50	number of training examples in one forward / backward pass
number of epochs	[20, 60]	10	40	maximum number of iterations

Table 2: Neural network parameters for both HOG and CNN features



(a) BER on the test set with respect to the number of units M (HOG features)

(b) BER on the test set with respect to the number of units M (CNN features)

Figure 3: Impact of number of units M on the BER of the test set

wish to determine M that achieves the best performance (lowest BER). Therefore, M is calculated with cross-validation (CV).

The NN has also numerous hyper-parameters such as the learning rate, the batch size and the number of epochs. The meaning of each parameter is given briefly in Table 2[5]. Since the search space for these parameters is very big, we do not perform CV for each one of them because it is time expensive. Instead, we try to optimize these parameters manually (grid search) by aiming at minimizing the BER on a fixed test set. We use both HOG and CNN features and get the same parameter values. The results are tabulated in Table 2.

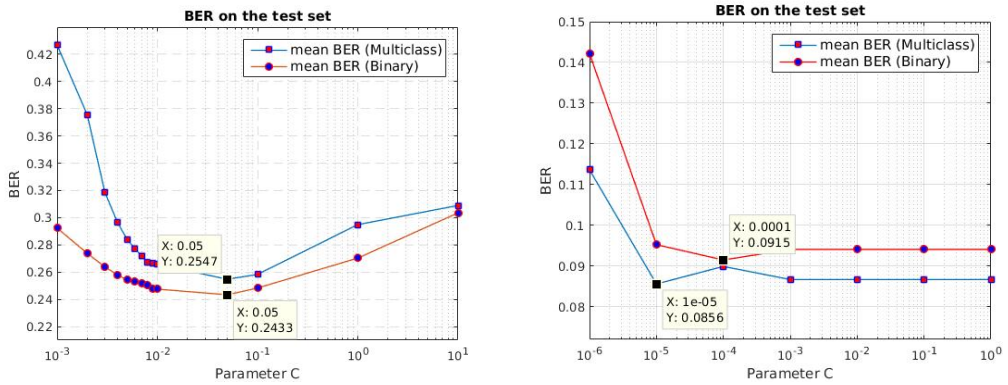
The provided NN-based classifier works as follows. We set up the NN by providing the number of units in the input, the output and the hidden layer. We also set the hyper-parameters according to Table 2, normalize $\mathbf{X}_{train}$ and pass it to the NN. As soon as the NN is trained, we normalize $\mathbf{X}_{test}$ using μ_{train} and σ_{train} and pass it to the NN in order to get predictions. The prediction for each test data point $\mathbf{x_i}$ is done by getting the most likely class, i.e. by choosing the output unit with the highest value.

Now, we perform 5-fold CV in order to find M (number of units in the hidden layer) that minimizes the BER on the test set. We vary M in the interval of $[10, 100]$ and $[10, 50]$ with step 10 when HOG and CNN features are used respectively. We limit the upper value of $M_{max,HOG} = 100$ and $M_{max,CNN} = 50$, since an increase in M leads to a big increase in the training time of the NN. This procedure is followed both for binary and multiclass classification. The results for HOG and CNN features are depicted in Fig. 3(a) and 3(b) respectively. The optimum values for M are presented in Table 4 and the respective achieved BER on the test set in Table 3.

4 Image Classification with SVM

This section presents an SVM-based image classifier. SVMs are inherently binary classifiers but can be turned into multiclass classification techniques by a variety of methods. The most common are the *one-versus-all* and the *one-versus-one* methods. In our implementation, we use the *one-*

[5]For detailed information regarding these and other parameters (e.g. `activation_function` and `weightPenaltyL2` not tested due to lack of time), please refer to the Deep Learn Toolbox documentation.



(a) BER on the test set with respect to C (HOG features)

(b) BER on the test set with respect to C (CNN features)

Figure 4: Impact of number of parameter C on the BER of the test set

versus-one method because it is faster compared to the *one-versus-all* method [5]. This method uses $K \cdot (K-1)/2$ binary classifiers (also called learners), where K is the number of classes (4 in our case). Finally, classification is done by running all binary classifiers and choosing the class with the highest prediction score. More information on multiclass classification can be found in [6].

Our SVM-based implementation uses templates and functions from the *Statistics and Machine Learning* Matlab toolbox. More concretely, we use the `templateSVM` in order to specify SVM hyper-parameters such as the kernel function, the polynomial order, the regularization parameter C (similar to λ in ridge regression) and the misclassification cost[6]. In order to train our classifier, we use the `fitcecoc` function that, by default, fits multiclass models using SVM. This function takes as input the normalized $\mathbf{X}_{train}$, the training labels $\mathbf{y}_{train}$ and the parameters specified in the `templateSVM` and returns a trained classifier. The prediction on the test set is performed using the `predict` function that takes as input the normalized $\mathbf{X}_{test}$ and the trained classifier and returns predictions on the test set, i.e. labels for each $\mathbf{x_{i,test}}$.

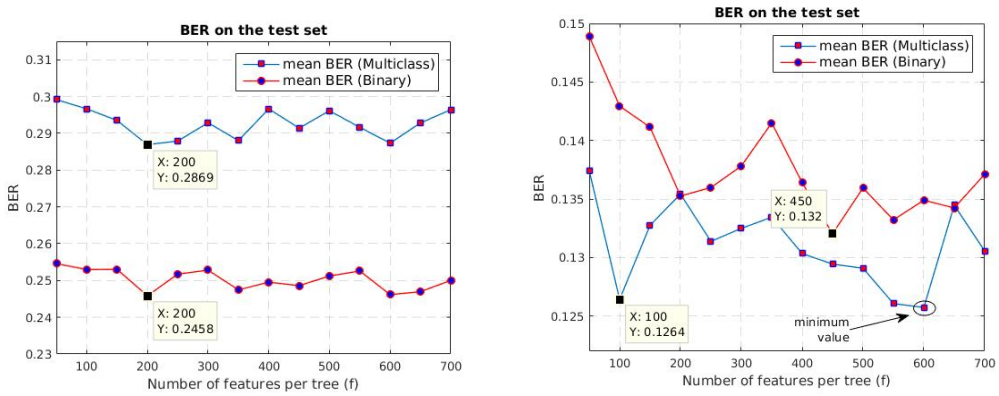
First, we keep the default value $C = 1$ and try linear, gaussian and polynomial (order: 2, 3 and 4) kernel functions while aiming at minimizing the BER. This procedure is followed both for HOG and CNN features. In all our simulations we take into account the performance (BER) - computational time tradeoff. We find that the linear kernel performs very well for both type of features and achieves a reasonable computational time - performance tradeoff. Then, we fix the kernel function to be linear and perform 5-fold CV to determine the best value for C with respect to BER minimization. In case of HOG features, we vary C in the interval of $[10^{-3}, 10]$ with 12 points in between. However, C is varied in the interval of $[10^{-6}, 1]$ with 5 points in between when CNN features are used (less points than HOG as the CV is time expensive with CNN). The results are depicted in Fig. 4(a) and 4(b) for HOG and CNN features respectively. The minimum BER on the test set can be found in Table 3 and the respective value for C in Table 4.

5 Image Classification with Random Forests

This section presents a RF-based classifier. Definitions, examples, advantages and disadvantages of RF can be found in [7]. The most common hyper-parameters of RF-based classifiers are (i) the number of trees that should be built, (ii) the number of features used for each tree and (iii) the tree depth.

Our RF-based implementation uses functions from the *Statistics and Machine Learning* Matlab toolbox instead of the *Piotr's* Toolbox (see *Appendix*). More concretely, we use the `TreeBagger` function in order to train our classifier. This function takes as input the normalized $\mathbf{X}_{train}$, the training labels $\mathbf{y}_{train}$ and parameters that specify the number of trees T and the number of features f per data point $\mathbf{x}_i$ (`TreeDepth` not included in `TreeBagger`). Our implementation uses also sampling with replacement (option: `SampleWithReplacement`). The prediction is done using the `predict` function that takes as input the trained classifier and $\mathbf{X}_{test}$ and returns labels for each $\mathbf{x_{i,test}}$.

[6] We do not experiment with the `Cost` parameter due to lack of time. The default value is 1.



(a) BER with respect to number of features f. HOG features used and $T = 100$

(b) BER with respect to number of features f. CNN features used and $T = 100$

Figure 5: BER with respect to number of features f ($T = 100$)

When working with RF-based classifiers, it is important to select the value of each parameter/hyper-parameter in a way that provides sufficiently low correlation with adequate predictive power [7]. For this reason, we perform 5-fold CV to get the optimum values for both T and f with respect to BER minimization. First, we start with $f = 100$ and vary T in the interval of $[50, 250]$ with step 25 but use only CNN features[7] (plot not included due to space). We do not use more than 250 trees as we take into account the performance - computational time tradeoff. The minimum value is $BER_{CNN,min} = 0.1256$ for $T = 175$ and $f = 100$. However, we choose to keep $T = 100$ with $BER_{CNN} = 0.1264$, since we get similar performance by using a simpler model (less than 1% difference with respect to 175 trees). Then, we set $T = 100$ and vary f in the interval of $[50, 700]$ with step 50. The mean BER on the test set is depicted in Fig. 5(a) and 5(b) for HOG and CNN features respectively for both binary and multiclass classification. In case of multiclass classification with CNN features, we get $BER_{CNN,min} = 0.1257$ for $f = 600$. However, we choose to keep again the simplest model which gives $BER_{CNN} = 0.1264$ for $f = 100$. The performance results the RF-based classifier are tabulated in Table 3 and its respective parameters in Table 4.

6 Feature engineering with SVM

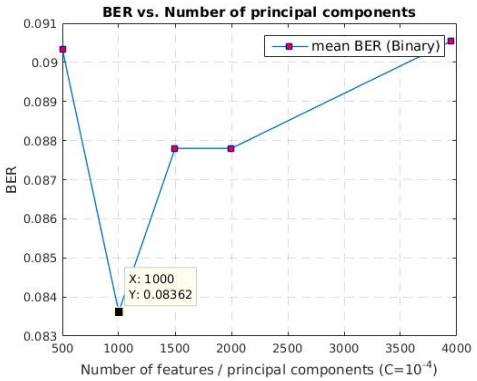
Now, we apply methods described in Section 2.2 for the model that gives the best performance and aim at further reducing the BER. From the above analysis and Table 3, we conclude that the SVM-based classifier with CNN features performs the best for both binary and multiclass classification. Hence, we focus only on SVM and CNN features and try to further increase performance. First, we remove features that display constant variance in the 99% of their data. This operation results to a significant reduction of all but 1163 features (CNN). We test the new data set and receive worse performance ($BER \simeq 0.15$). In addition, we remove outliers but still do not notice any performance improvement. Also, merging the HOG and CNN features into one $\mathbf{X}$ does not reduce the BER. Last but not least, we apply PCA and get performance for different combinations of the number of principal components d and C. We conclude that the best C is 10^{-4} and 10^{-3} for binary and multiclass classification respectively. Fig. 6(a) and 6(b) show how the BER changes with respect to d. We notice a performance improvement for both binary and multiclass classification. This means that a lower BER is achieved by using a simpler and faster to train model (less features). The results and the value of the hyper-parameters are presented in Section 7.

7 Results

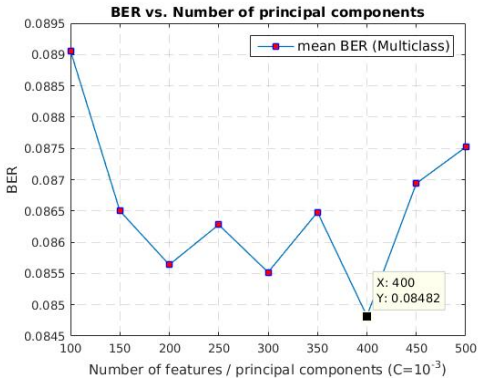
We summarize our findings from the classification methods we analyzed in Table 3. The SVM-based classifier with PCA gives the best performance for both binary and multiclass classification with CNN features. The hyper-parameters that achieve the respective performance per classifier are listed in Table 4.

We now test the performance of all models with the best combination of hyper-parameters on 10 different train/test sets using only CNN features. The mean BER and the confidence interval for

[7] Simulations with HOG features show that the achieved BER is higher compared to CNN features.



(a) BER vs. number of features/principal components d for $C = 10^{-4}$ (binary classification)



(b) BER vs. number of features/principal components d for $C = 10^{-3}$ (multiclass classification)

Figure 6: BER vs. number of features/principal components d

	HOG features		**CNN features**	
	Binary	Multiclass	Binary	Multiclass
Baseline model	0.5	0.75	0.5	0.75
Neural network	0.241	0.272	0.107	0.103
SVM (linear)	0.243	0.255	0.092	0.087
Random forest	0.246	0.287	0.132	0.126
SVM (linear) with PCA	*not tested*		0.084	0.085

Table 3: Comparison of best BER per model (BER - computational time tradeoff taken into account)

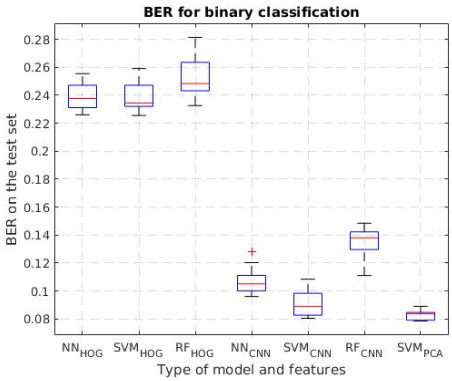
	HOG features		**CNN features**	
	Binary	Multiclass	Binary	Multiclass
Neural network	$M = 100$	$M = 70$	$M = 40$	$M = 30$
SVM (linear)	$C = 0.05$	$C = 0.05$	$C = 10^{-4}$	$C = 10^{-5}$
Random forest	$f = 200$ $T = 100$	$f = 200$ $T = 100$	$f = 450$ $T = 100$	$f = 100$ $T = 100$
SVM (linear) with PCA	*not tested*		$C = 10^{-4}$ $d = 1000$	$C = 10^{-3}$ $d = 400$

Table 4: Hyper-parameters that achieve the BER per model

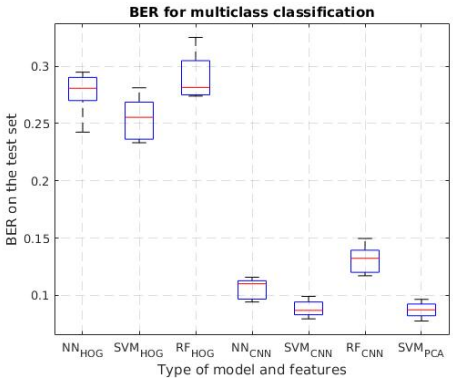
	CNN features			
	Binary	Conf. interval (95%)	Multiclass	Conf. interval (95%)
Baseline model	0.5		0.75	
Neural network	0.1072	[0.0999, 0.1146]	0.1069	[0.1009, 0.1129]
SVM (linear)	0.0908	[0.0842, 0.974]	0.0884	[0.0836, 0.0931]
Random forest	0.1342	[0.1259, 0.1425]	0.1316	[0.1229, 0.1402]
SVM (linear) with PCA	**0.0829**	[0.0802, 0.0855]	**0.0868**	[0.0824, 0.0913]

Table 5: Mean BER and confidence interval (95%) for optimum hyper-parameters per model. Only CNN features used.

each model are presented in Table 5 and visualized in Fig. 7. The baseline is not included in Fig. 7 because it would ruin the scaling and granularity of the y-axis. However, it is included in Table 5. For completeness, we mention that $BER_{train,binary} = 0.037$ and $BER_{train,multiclass} = 0.0241$ for our best model (SVM with PCA).



(a) Box plots for binary classification: 1. NN with HOG 2. SVM with HOG 3. RF with HOG 4. NN with CNN 5. SVM with CNN 6. RF with CNN 7. SVM with CNN and PCA



(b) Box plots for multiclass classification: 1. NN with HOG 2. SVM with HOG 3. RF with HOG 4. NN with CNN 5. SVM with CNN 6. RF with CNN 7. SVM with CNN and PCA

Figure 7: Box plots for binary and multiclass classification

Performance tradeoffs

From the methods we analyzed, RF is the fastest and SVM the slowest one. In case computational time is more important than performance, an RF-based classifier should be chosen. Moreover, we believe that one should use PCA since it both reduces the dimensionality of $\mathbf{X}$ and speeds up the training time while achieving performance similar to the one obtained when all features are used.

Observations

Along with the results, we present some observations regarding our analysis.

- **Binary vs. Multiclass Classification:** We believe that we add *noise* to our data set by considering pictures with different objects to belong to the same class. By doing so, we may destroy patterns (edges, contours, etc.) that our classifiers may *learn* while training, which may explain the worse performance of binary against multiclass classification with CNN features. However, we notice that SVM with PCA overcomes this problem. Finally, binary classification performs better than multiclass, when HOG features are used.
- **Where do the most mistakes/misclassifications happen:** By analyzing the outcome of our methods, we observe that the majority of misclassifications happen between *horse* and *other*. For example, images 4769 and 4782 depict horses but are classified as others (the head of the horse is missing from the picture which may confuse the classifier).

8 Conclusion

Our analysis leads us to choose the SVM-based method with CNN features and PCA as the best classifier for both binary and multiclass classification. Our predictions are reported in the `pred_binary.mat` and `pred_multiclass.mat` files.

Appendix: Experimenting with other methods

We plot the correlation coefficients of HOG features with the output and notice a periodicity in the diagram. However, the performance is worse after removing features with repeated correlation coefficients.

Piotr's toolbox: `forestTrain` and `forestApply` methods

`TreeBagger` does not provide any `TreeDepth` option. For that reason, we tried the `forestTrain` and `forestApply` methods from Piotr's toolbox and experimented with the `TreeDepth` option. However, we did not notice any performance improvement ($BER_{CNN,best} = 0.185$ for $depth = 50$) and decided to use `TreeBagger`.

9 Acknowledgments

We would like to thank Emtiyaz Khan and the TAs for their advice and help throughout the implementation of the project. The code was written by Giannakopoulos Athanasios and Kyritsis George.

References

[1] Navneet Dalal and Bill Triggs, *Histograms of Oriented Gradients for Human Detection*.

[2] OverFeat: Object Recognizer, Feature Extractor, `http://cilvr.nyu.edu/doku.php?id=software:overfeat:start`

[3] Kay H. Brodersen et. al, *The balanced accuracy and its posterior distribution*. 2010 International Conference on Pattern Recognition.

[4] Christopher M. Bishop, *Pattern recognition and machine learning*. Springer, p.257.

[5] Ryan Rifkin, *Multiclass Classification*. MIT Feb. 2008.

[6] `http://ch.mathworks.com/help/stats/fitcecoc.html#bufm0tv`.

[7] Ned Horning, *Random Forests: An algorithm for image classification and generation of continuous fields data sets*.