PCML: Project-II by Group BEIJING

Fei Mi
fei.mi@epfl.ch

Yumeng Hou
yumeng.hou@epfl.ch

December 21, 2015

1 Introduction

In this project, our goal is to apply various machine learning methods to real-world classification problems. The task is to recognize whether specific objects present completely or partially in the images. There are three categories of interested objects: Airplane, Car and Horse, labeled as $y = 1, 2, 3$ respectively. As for images that contain none of these objects, we classify them as "Others" and label $y = 4$. Each image, either color or grayscale, is of size $231 * 231$ and has the associated label y. We study, implement and test several state-of-the-art classifiers trained on this dataset. In this report, we manifest and compare different methodologies, and discuss the performance of different approaches.

2 Literature Review

Significant progress has been made in the area of object recognition due to its practical applications such as CV and CBIR. For a long time, segmentation was the leading option for recognition by partitioning the image and searching for a matching unit [1]. But since there is not always a generic solution for hierarchical segmentation and restriction exist, a new approach was proposed to do localization through the identification of an object [5]. As an object can be located at any position and scale, to search everywhere in the huge visual space is computationally expensive. Using weak classifiers such as HOG to do sliding windows turns preferred. Afterwards, such features are fed to a classifier, typically a support vector machine (SVM) which proves to be a successful technique for classification tasks. It's mostly due to HOG features being discriminative and SVM classifier being margin-based linear [10].
In the complex context, for example where objects appear at different scales and orientations, low-level features become ineffective. More advanced features and descriptors are necessary to capture the semantics of the scene, which gives way to deep neural networks. Among various neural approaches, the convolutional neural network (CNN) proves mostly efficient in face recognition [8]. Several architectures have been proposed for using CNNs, especially for computer vision tasks. Among them, CaffeNet and GoogLeNet are promising. Caffe is one of the most popular libraries for deep learning and CNN in particular. It is easily customizable through configuration files, extendible with new layer types, and provides a very fast CNN implementation. It also provides C++, Python and MATLAB APIs [6]. GoogLeNet's [14] main peculiarity is the use of inception modules. In very concise terms, inception modules reduce the complexity and thus multiple filters can be used in parallel at different resolutions. It also employs auxiliary classifiers connected to intermediate layers.Moreover, in a most recent research, a novel method (HCNN) to improve CNN with HOG is proposed [15]. The experimental results show that HCNN outperforms traditional CNN for object classification with fooling images.

3 Dataset and Feature Description

In this section, we will first describe our dataset, then introduce the features and their characteristics, based on which we apply different preprocessing, visualization and model-training approaches accordingly.

3.1 Dataset Statistics

Our dataset consists of 6000 images of size $231 * 231$, either color or grayscale, and a train.mat containing the HOG feature, CNN feature and an associated label y for each image. The HOG features are generated with Piotr's toolbox, in a dimension of $13 * 13 * 32$. And the OverFEAT ImageNet CNN Features are of $6 * 6 * 1024$,

trained from the huge ImageNet database. There are four classes in total, differentiated by label y. Interested objects include Airplane, Car and Horse, labeled as $y = 1, 2, 3$ respectively. As for images that contain none of these objects, we classify them as "Others" and label $y = 4$. Following a statistically exploratory data analysis, we get the idea that there are 964 images in class Airplane, 1162 in Car, 1492 in Horse and the rest 2382 in Others. There is an obvious imbalance problem in this dataset, which stays as a concern in choosing suitable evaluation criteria and sampling training data. We will clarify these in the following sections.

3.2 Histogram of Oriented Gradients (HOG)

As a descriptor computed on a dense grid of uniformly spaced cells and employs overlapping local contrast normalization for improved accuracy, HOG is widely used for object detection. As illustrated in Figure 1, basically, it decomposes an image into small squared cells, computes an histogram of oriented gradients, normalizes the result using a block-wise pattern, and return a descriptor for each cell [4], Hence, HOG is robust towards illumination and background clutter, and has been popular used as object representation.

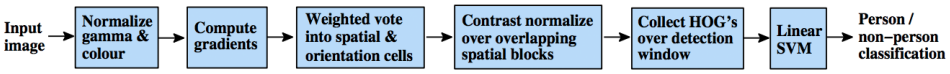

Figure 1: A pipeline of HOG feature extraction and application in a conventional SVM classifier [4]

HOGs can be used as an image window descriptor for object detection, for example by means of SVM. However, HOGs are scale- and rotation- invariant. Object localization requires the evaluation of the similarity of a reference HOG to the HOG computed in each and every possible placement of window. Thus, we would like to uniform the scale and rotation while training HOGs for better prediction. Or try to conduct independent detection, by performing explicit and exhaustive consideration of all the search dimensions, which makes the computation cost expensive.

3.3 Convolutional Neural Networks (CNN)

OverFeat is a Convolutional Network-based image features extractor and classifier, where different data augmentation schemes can be implemented to improve classification results. CNN have showed success in achieving translation invariance for many image processing tasks. The main advantage of it is that the entire system is trained end to end, from raw pixels to ultimate categories, thereby alleviating the requirement to manually design a suitable feature extractor. The main disadvantage is their ravenous appetite for labeled training samples. Evolving from the multilayer perceptron (MLP), CNN adopt the idea to process architecture based on a large number of layered and massively interconnected simple units. Neuron, the basic processing unit, is very simple, and computes the output activation by comparing the weighted sum of its input with a threshold and applying a suitable nonlinearity.
Different form MLP, CNN neurons have limited receptive fields. And all neurons of a layer are identical to one another, except for their receptive fields, sharing the same weights. These constraints reduce sharply the number of free parameters to learn. As the name suggests, the $(n+1)^{th}$ layer computes (before the nonlinearity) a spatial convolution of the outputs of the n^{th} layer. Therefore, it extracts some basic features of the image which are passed on to the next layer for further processing [2].

3.4 Visualization of HOG Features

For basic visual observation, we use Piotr's $hogDraw()$ function to visualize the extracted HOG features. Figure 2(a) shows the visualization of HOG features according with the exact image. It can be seen that under different scale and rotation situation, features could get diverse even within the same class. For example, we could have two airplane image with different orientations or scale.

3.4.1 Preprocessing to Produce Fine-Grained Labels based on HOG Features

In preprocessing, we try to pre-cluster the HOG features based on the orientation of interested objects. Basically, the idea is to distribute subsets with domination of one kind of objects in a specific orientation. In detail, we have

two subclasses for airplane and car, three subclass for horse. Figure 2 demonstrates our pipeline of partitioning training set and results. Af first, we split the whole set into subsets of identical objects according to label y. Then for each subset, we pick up several typical images with clear orientation information as seeds, apply basic SVM. As a result, we harvest a more fine-grained class label and each corresponds to one figure with specific orientation. However, as we will elaborate in Section 5, using those fine-grained labels still cannot help HOG perform better and it leads us to pursue optimization using only CNN features.

Figure 2: Pipeline of partitioning training set into uniform subsets (a)Visualization of HOG Features

4 Methodology

4.1 Evaluation Metric

Balanced Error Rate (BER) is the arithmetic mean of the misclassification rate in each individual class. BER is suitable for binary and multi-class prediction problem. It also fits the situations where the distribution of objects is biased among classes, which addresses the problem of imbalance within our dataset. Thus in this project, we use BER as the evaluation measure. BER is computed as Equation 1, where C is the number of classes, N_c is is the number of examples in class c, y_n is the ground truth for sample n and $\hat{y}_n$ is its prediction.

$$BER = \frac{1}{C}\sum_{c=1}^{C}\left[\frac{1}{N_c}\sum_{n=1}^{N}(y_n = c)(y_n \neq \hat{y}_n)\right], \tag{1}$$

We will focus on achieving minimum BER as our evaluation metric, and all the error rates reported in the following sections are computed though five-fold cross validation.

4.2 Support Vector Machine (SVM)

In machine learning, a Support Vector Machine (SVM) is a very commonly used discriminative supervised learning model which learns a “hyperplane” that separates the target class from the rest, so it is also called a “large-margin Machine”. For a binary classification task, an SVM model is a representation of the instances as points in space, mapped so that the instances of the target class are divided from the other class by a hyperplane as much as possible. New examples are then mapped into that same space and predicted to belong to a class based on which side of the hyperplane they fall on. In addition to performing linear classification, SVMs can also efficiently perform a non-linear classification using kernel tricks by mapping inputs into high-dimensional feature spaces. Multiple kernel functions could be applied for a nonlinear SVM, and the one adopted in this thesis is radial basis function (RBF) kernel. The RBF kernel function is defined below in Equation 2 with the distance of $\boldsymbol{x}$ and $\boldsymbol{x'}$ in high-dimensional feature space is denoted by $K(\boldsymbol{x}, \boldsymbol{x'})$ and γ is a free parameter that we could modify.

$$K(\boldsymbol{x}, \boldsymbol{x'}) = exp\left[-\frac{||\boldsymbol{x} - \boldsymbol{x'}||^2}{2\gamma^2}\right] \tag{2}$$

We use two variants of SVMs, linear SVM and SVM with the radial basis function (RBF) kernel, on the LIBSVM package [3]. The parameter C of the linear SVM and the parameters pair (C, γ) of the SVM with RBF kernel are tuned via 5-fold cross-validation on the entire training set.

4.2.1 Computational Problem

The mainly cause of the computational problem of SVM, especially for kernel-based SVM is that it requires on the order of n^2 computations for training and order of $n \times d$ computations for classification, where n is the number of training examples and d the input dimension. For the two features we have, the dimensionality of CNN features is much higher than HOG features. Due to our computational limitations, we only run SVMs with linear and RBF kernel for HOG features, linear SVM for CNN feature, yet SVM with RBF for CNN features. We will mainly illustrate the process of running SVMs using HOG features in the following part of this section.

4.2.2 Hyperparameter Tuning using HOG Features.

Tuning the hyperparameters is a vital step in best practice of applying SVMs. In the following part, we will discuss the parameter tuning issues of applying SVMs with different kernels.

SVM with Linear Kernel For SVM with linear kernel, we need only to tune the hyperparameter C, which is essentially a regularization parameter. It controls the trade-off between achieving a low error on the training data and minimizing the norm of the weights. It is analogous to the ridge parameter in ridge regression. The C parameter tells the SVM optimization how much you want to avoid misclassifying each training example. For large values of C, the optimization will choose a smaller-margin hyperplane and it could lead to overfitting. Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplanes and could lead to underfitting.

For tuning the hyperparameter of C of SVM with linear kernel, we use the basic line search method, that is testing a range of possible C and choose the one with the best performance with five-fold cross validation. The left part of figure 3 presents the results of tuning parameter C for linear SVMs.

SVM with RBF Kernel For SVM with RBF Kernel, we have one more hyperparameter γ to be tuned. In this setting, the separating surface will be based on a combination of bell-shaped surfaces centered at each support vector. The width of each bell-shaped surface will be inversely proportional to γ. If this width is smaller than the minimum pair-wise distance for your data, you essentially have overfitting. If this width is larger than the maximum pair-wise distance for your data, all your points fall into one class and you don't have good performance either. So the optimal width should be somewhere between these two extremes and being data dependent as well.

The traditional method utilizes a two dimensional uniform grid of points (grid search) in the space and find a combination that gives the least value for some estimate of generalization error. This method guarantees to find a good solution, yet it's extremely computationally intensive. So we use an efficient heuristic method proposed and proved by [7] for finding a good hyperparameter more efficiently. Assume both parameters are in the space of log_2, then the heuristic is given by:

1. Search for the best C of Linear SVM and call it $\bar{C}$.

2. Fix $\bar{C}$ from step 1 and search for the best (C, γ) satisfying $log\gamma^2 = logC - log\bar{C}$ using the RBF kernel.

Since we utilize and compare SVMs with both linear and RBF kernel, the above heuristic is extremely computational efficient. For traditional grid search method, the time complexity of grid search for searching hyperparameter C and γ is $p \times q$ if q and q are the number of values we search for C and γ respectively. The above heuristic could reduce the complexity to be $2 \times p$ for tuning linear and RBF kernel all together.

The left part of Figure 3 gives the results of tuning parameter C for SVM with linear kernel. The evaluation metric used is balanced error rate (BER) as given in the previous section. We could see that the best C for linear SVM is 2^-12, that is the $\bar{C}$ above. When we tune the parameter (C, γ) for SVM with RBF kernel, we only using one γ corresponds to each C that follows the condition of the second step of the heuristic above, so that we could get the results in the right part of Figure 3.

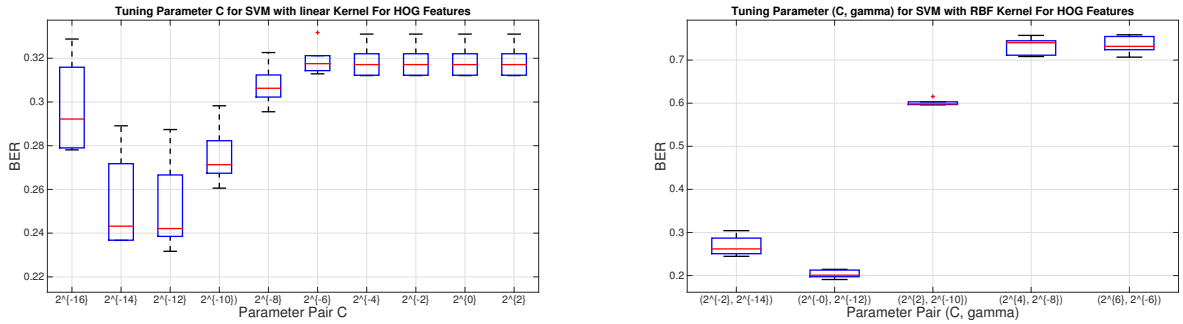

Figure 3: Left: Tuning hyperparameter C of Linear SVM for HOG features. Right:Tuning hyperparameter (C, γ) for SVM with RBF kernel for HOG features using the heuristic above.

4.3 Neural Network

A multilayer perceptron (MLP) is a feedforward artificial neural network model that maps sets of input data onto a set of appropriate outputs [11, 12]. An MLP consists of multiple layers of nodes in a directed graph, with each layer fully connected to the next one. Except for the input nodes, each node is a neuron (or processing element) with a nonlinear activation function. In terms of optimization, MLP utilizes a supervised learning technique called backpropagation for training the network. The multilayer perceptron consists of three or more layers (an input and an output layer with one or more hidden layers) of nonlinearly-activating nodes and is thus considered a deep neural network. Since an MLP is a fully connected network, each node in one layer connects with a certain weight w_{ij} to every node in the following layer.

The implementation of the MLP is based on the deep learning toolbox for Matlab as given, and we add another activation function as introduced in later section. Some basic optimization parameters need to be set, like the learning rate for network weight and number of full sweeps through data during training (number of epochs). We fix the learning rate to be 2, and observed that the model learned converge quite fast, all less than 10 iterations. So we set the number of full sweeps through data between 15 and 20 to ensure training convergence. Besides the optimization parameters, the structure of the network is even more crucial that we will stress in the following section.

4.3.1 Structure Design of MLP

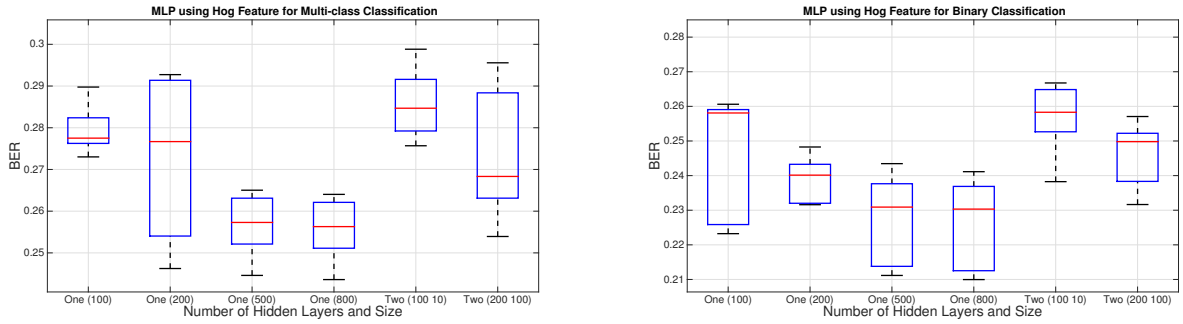

Figure 4: Tuning the number of hidden layers and number of neurons in each hidden layer for HOG features through five-fold cross validation.

The Structure of ML need to be first designed through by varying the number of hidden layers and number of neurons in each hidden layer. Figure 4 illustrate the results obtained by varying the network structure using HOG features for both multi-class classification and binary classification task, and Figure 5 illustrate the obtained for using CNN features. For all four classification results below, we have four kinds of networks with one hidden layer with number of hidden neurons varying from 100 to 800, besides, we also tried two settings of network structures with two hidden layers. Several observation could be drawn from the results below:

1. For both multi-class and binary classification task, utilizing CNN features could give us a superior performance compared with HOG features.

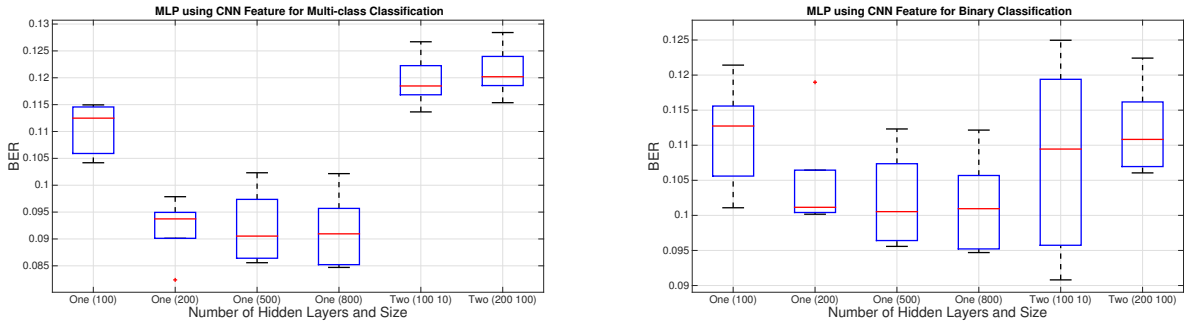

Figure 5: Tuning the number of hidden layers and number of neurons in each hidden layer for CNN features through five-fold cross validation.

2. Using two hidden layers doesn't give a superior performance for our tasks. In principle, a deeper network could achieve better performance for matching inputs and labels, yet deeper network require much more data to train the model.

3. One hidden layer, with enough hidden units (500 or 800), bears enough representation power for our task. As we could observe from the four figures above, one hidden layer with 500 hidden neurons is a excellent choice balancing performance and computational efficiency.

Adopted Setting So Far As stated in the previous results, CNN features are more powerful than HOG features for our task, so we will focus on optimize the models built on top of CNN features. Besides the generally network structure that we will utilize is the MLP with one hidden layer with 500 hidden neurons. In the following three sub-sections, will will applied three commonly used tricks to design special structures for MLP.

4.3.2 Activation Function

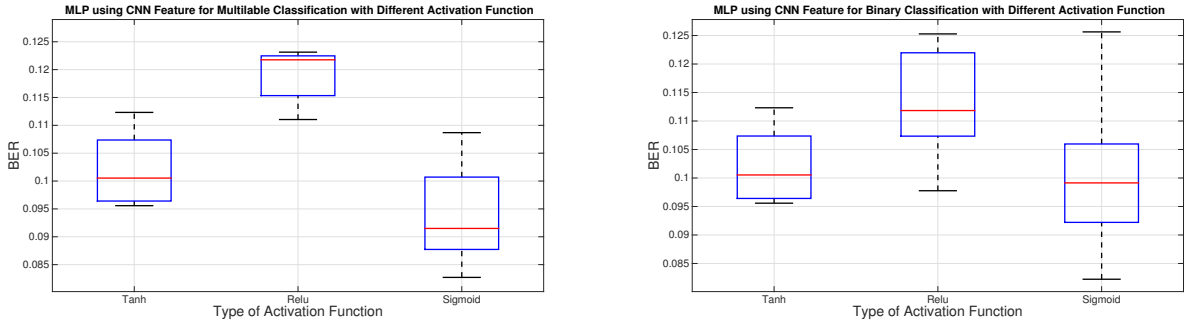

Figure 6: Apply different activation functions for the adopted architecture proposed above (Left: multi-label classification; Right: binary classification).

The first variation it to assignment different activation functions to the hidden neurons. The default hidden layer activation function we use is tanh, and the results illustrated in Figure 4 and Figure 5 adopt the tanh hidden layer activation function. However, there are some other types of commonly used activation function with different properties.

Sigmoid activation function Sigmoid activation function is another common choice and it's studied wide during our course. It is an S-shaped (sigmoid) curve, with output in the range (0,1) while the output of tanh is in the range (-1,1).

Relu activation function The Relu activation function is defined in Equation 3 with output in the range (0,1). It's also called softplus function whose derivative is the logistic function. Actually, it's the most popular

activation function for deep neural networks used so far as stated by [9]. We implemented the Relu activation function within the setting of given Matlab deep learning toolbox.

$$f(x) = ln(1 + e^x) \tag{3}$$

As we could observe from Figure 6, Sigmoid activation function achieves the best performance compared with tanh and Relu activation functions for our task, while Relu activation actually performs the worse. In this sense, we fix the activation function to be Sigmoid when we try to apply the following tricks, weight decay and dropout.

4.3.3 Weight Decay

In order to effectively limit the number of free parameters in our MLP so as to avoid over-fitting, we could regularize the weight of our MLP so that it penalizes large weights and effectively limits the freedom in our model. The idea of weight decay is using a regularization term so that the weight decays in proportion to its size at each optimization step. We have witness the usefulness and necessity of regularization from both course and project exercise. The parameter to be tuned in this method is the regularization parameter which penalizes the L2 norm of the size of the network weights..

4.3.4 Dropout

Dropout is another technique for addressing overfitting problem proposed by [13]. The key idea is to randomly drop units (along with their connections) from the neural network during training. This prevents units from co-adapting too much. It could significantly reduces overfitting and gives some improvements over other regularization methods. The major parameter to be tuned in this method is the fraction of nodes to be dropped out.

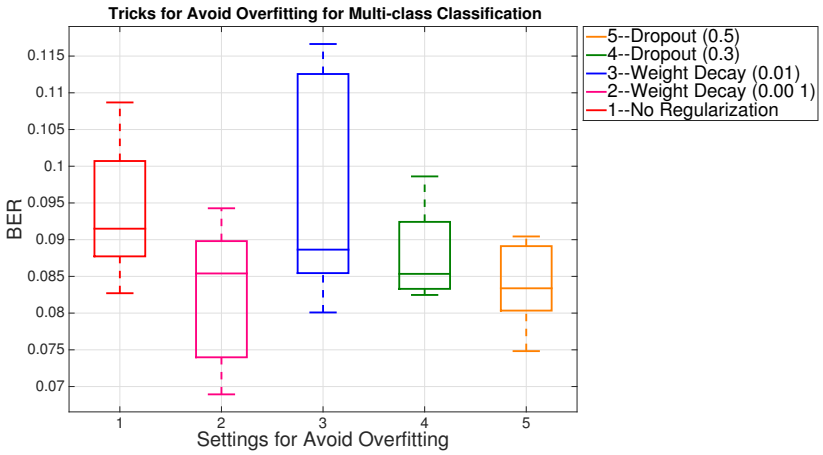

Figure 7: Results of adding weight decay and dropout

We could see from Figure 7 that both weight decay and dropout could boost the performance slightly. For weight decay, the model with regularization parameter 0.01 performances better than with regularization parameter 0.001, while for dropout, the model with 0.5 dropout fraction performances better than with dropout fraction 0.3. In generally, the best alternative is the dropout with fraction 0.5, and we choose it to be our optimal setting which improves the previous best setting (with out regularization) by around 1% BER.

Optimal network setting Until now we have draw a conclusion about the model we are going to use for classification on the testing set. We will only utilize CNN features and build a MLP with one hidden layer of 500 hidden neurons. Besides, both the activation of hidden layer and output layer are sigmoid activation with dropout fraction 0.5. The predictions we give for the testing data are also based on this setting.

4.4 Alternative Implementation of Binary Classification

Projected from Multi-class Classification Problem Previously, the binary classification problem is formulated as classical binary classification setting, that is, both training and test labels contains only two classes.

We observe from the results in Figure 6 that the performance of multi-class classification is always slightly better than binary classification. So we consider to directly project or convert the prediction given by multi-class classification to the results of binary classification. That is for the predictions given by multi-class classification, label 1, 2, 3 will be projected to label 1 of binary classification and label 4 projected to label 0 of binary classification. From the results in the Figure 8, we could see that generating binary predictions from multi-class classification could achieve an even better performance as we expected, with around 0.7% improvement in terms of BER.

Figure 8: Different ways to produce the results of binary classification

Best Prediction Performance With different settings and variations we studies in the previous sections, we have our optimal average BER to be 8.45% for multi-class classification and 7.86% for binary classification through five-fold cross validation.

5 Conclusion

In this project, we accomplished image classification task with binary and multi-class prediction based on HOG and CNN features. We applied two variants of SVM, linear SVM and SVM with RBF Kernel. We tuned the hyperparameters of SVM with RBF kernel using an efficient heuristic. We also implemented multilayer perceptron (MLP), a feedforward artificial neural network model on both features for both type of classification tasks. With optimized parameters and network structures, we also tried various activation functions and different ways to avoid overfitting. Through our testing and exploration, the optimal setting only utilizes CNN features and build a MLP with one hidden layer of 500 hidden neurons. Besides, both the activation of hidden layer and output layer are sigmoid activation with dropout fraction 0.5. Such setting achieve average BER of 8.45% for multi-class classification and 7.86% for binary classification.

References

[1] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik. Contour detection and hierarchical image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 33(5):898–916, 2011.

[2] Marco Castelluccio, Giovanni Poggi, Carlo Sansone, and Luisa Verdoliva. Land use classification in remote sensing images by convolutional neural networks. *arXiv preprint arXiv:1508.00092*, 2015.

[3] Chih-Chung Chang and Chih-Jen Lin. Libsvm: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.

[4] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[5] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(9):1627–1645, 2010.

[6] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the ACM International Conference on Multimedia*, pages 675–678. ACM, 2014.

[7] S Sathiya Keerthi and Chih-Jen Lin. Asymptotic behaviors of support vector machines with gaussian kernel. *Neural computation*, 15(7):1667–1689, 2003.

[8] Steve Lawrence, C Lee Giles, Ah Chung Tsoi, and Andrew D Back. Face recognition: A convolutional neural-network approach. *Neural Networks, IEEE Transactions on*, 8(1):98–113, 1997.

[9] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[10] Yanwei Pang, Yuan Yuan, Xuelong Li, and Jing Pan. Efficient hog human detection. *Signal Processing*, 91(4):773–781, 2011.

[11] Frank Rosenblatt. *Perceptions and the theory of brain mechanisms*. Spartan books, 1962.

[12] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985.

[13] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[14] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2014.

[15] Tielin Zhang, Yi Zeng, and Bo Xu. Hcnn: A neural network model for combining local and global features towards human-like classification. *International Journal of Pattern Recognition and Artificial Intelligence*, page 1655004, 2015.