Project-II by Group Macau

Chen Liu
EPFL
chen.liu@epfl.ch

Mengjie Zhao
EPFL
mengjie.zhao@epfl.ch

Abstract

This report demonstrates the Project-II working results of group Macau in the 2015 Pattern Classification and Machine Learning (PCML) course. The task is to classify images based on given features. We firstly demonstrate our analysis to the provided feature sets. After that, our constructed classification models will be illustrated. We utilized the balanced error rate (BER) to evaluate models' performance and after estimating the testing performance of each model, we employed the best model to generate predictive outputs of provided testing instances.

1 Introduction

Our task in Project-II of the PCML course is to build a system which can identify the types of objects presenting in a 231×231 pixels image. The given dataset contains 4 types of images:{Car, Horse, Airplane, Others}. First of our task is to decide whether or not it is in the set {Car, Horse, Airplane} and the other is to classify the given image to a specific type. We constructed several classification models such as support vector machine and neural networks to solve this problem. Cross validation was carried out and multi-layer perceptron model using overFEAT features gives the best performance within both binary and multi-class prediction tasks. Hence we employ it to generate predictions on test data. The results are saved in $\mathbf{pred\_binary.mat}$ and $\mathbf{pred\_multiclass.mat}$ respectively.

2 Data description and feature analysis

The provided training data set contains 6000 images, each of them consists of 231×231 pixels. Images showing airplane, car and horse are assigned with label $1, 2$ and 3 respectively, and remaining images are labeled with 4. We do not need to abstract features from images by ourselves since two feature matrices $\mathbf{train.X\_cnn}$ and $\mathbf{train.X\_hog}$ are provided to help us building our classifiers. Before building classification models, we firstly tried to analyze the given features sets.

2.1 Histogram of Oriented Gradients (HOG)

According to Dalal's work in [1], we can divide one entire image into many rectangular patches. Within each patch, we calculate the gradient of each pixel which gives a weighted vote to an edge orientation histogram. By implementing this method, we can abstract each patch's features by the corresponding values in this histogram, which denote weighted orientations within this patch. In the classification red step, features that highly relate to the label are assigned with more weights whereas others are ignored by the classifier. HOG only demonstrates partial features. It can be easily inferred that the HOG features have modest performance, since it is sensitive to the location and orientation of the objects presenting in a image, even for same objects. Pictures below show an original image and its corresponding gradient distribution.

In the provided HOG feature matrix, each image corresponds to a 5408-by-1 dimensional vector. By analyzing the features, we found that every consecutive 169 features have rather similar patterns.

(a) Original Image

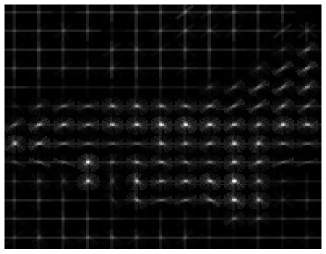

(b) Gradient Distribution

Figure 1: Original image and its gradient distribution

Since neighboring patches have similar oriented gradient distribution, and $169 = 13 \times 13$, $5408 = 169 \times 32$, we believe that when obtaining the HOG features, each image is divided into 13×13 cells and each cell contains a 32-bin histogram.

2.2 Features from Convolutional Neural Network (OverFeat)

The convolutional neural networks (CNN) have achieved remarkable performance in the field of computer vision. In Alex's work on ImageNet [2], he proposed a 8-layer deep neural network, including 5 convolutional layers and 3 fully connected layers, to classify 1.2 million images into 1000 categories. Section 3.3 will give a more detailed introduction to CNN and related neural network models.

The provided feature vectors of CNN are of 36864-by-1 dimensions. It is an indication that they are the outputs of the last convolutional layer, which produces 1024 features maps and each of them has a size 6×6. We noticed that most of given features are 0, hence, we believe that the activation function of neuron is the rectified linear unit (ReLU), i.e. $f(x) = max(0, x)$.

3 Constructed models

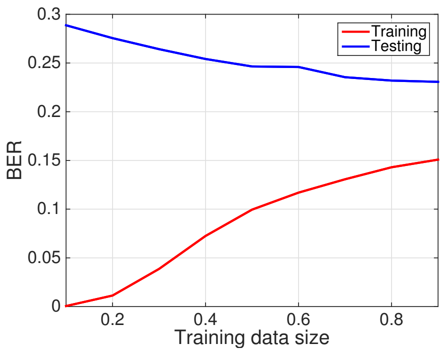
Within this section, several classification models will be demonstrated one by one. As the provided data set has 6000 images, there might be strong biases between samples of different type of images. Hence, we employed the balanced error rate (BER) to evaluate model performance. Also, in the following sections, all 'accuracy' denote balanced accuracy rate which equals to $1 - BER$.

In addition, to clearly demonstrate prediction results of our classifiers, besides outputting BER, our algorithms will also generate a validation matrix $\mathbf{M}$, where $\mathbf{M}_{ij}$ denotes the number of instance belonging to type i and classified as type j.

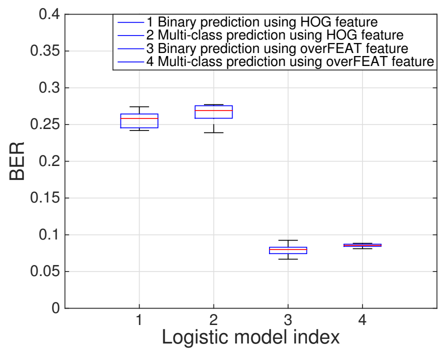
3.1 Logistic regression

Firstly, we applied logistic regression in the binary prediction task. However, for the 4-class prediction task, we need to construct a combined logistic regression model, which contains $C_4^2 = 6$ naive logistic regression models. Each of naive logistic regression models is responsible to decide the type of input image between two types. When a testing image arrives, it will be send to all the 6 naive logistic regression models. Then each model gives a prediction on the image and a voting process is carried out. As a result, the image will be assigned with the label that has the highest votes from all the 6 naive logistic regression models. Following figures demonstrate our findings.

Figure.2(a) demonstrates the training and validation BER alterations along with the growth of training data set, in the binary prediction task using naive logistic regression that utilizes HOG features. It can be observed that when more than 80% of provided datasets are split as training data, the BER performances becomes relatively stable. On the other hand, too big training set means a limited validation set, which will introduce large variance of the validation accuracy. Hence, we decided to use 5-fold cross validation in examining different learning models.



(a) Learning curve of the binary logistic regression using HOG features. We set 80% of data as training set and 20% of data as validation set.



(b) Performance of logistic regression in binary prediction and multi-class prediction tasks.

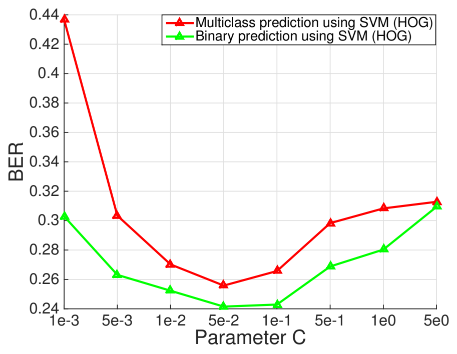
Figure 2: Learning curve and performance of logistic regression

Figure.2(b) displays BER performance of our logistic models from 5-fold cross validation, in both binary and multi-class prediction tasks. It can be observed that models using overFEAT features give much lower BER than that using HOG features, which means that overFEAT features contain more information from images than HOG features. This result meets our expectation in the feature analysis part.

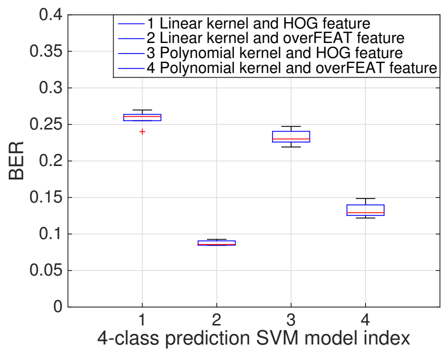
3.2 Support Vector Machine

Support vector machine (SVM) is a powerful classifier for solving binary classification problems. Defined in the feature space, one SVM discovers the largest margin between support vectors and the decision hyperplane. The learning strategy for a SVM is to maximize the margin and can be transformed into solving a convex optimization problem with the assistance of Lagrange multiplier. Combined with kernel functions, SVM can map the input into a high dimensional feature space, where the original linear inseparable problem may be more linear separable.

In the binary prediction task, we trained one SVM model to classify given images. In the multi-class prediction task, we carried out the same voting process in the combined logistic regression. In every experiment, all SVMs apply same type of kernel function. Following figures demonstrate our findings:



(a) Effects of tuning the parameter C, in both binary prediction and multi-class prediction. Numbers are arithmetic mean of results from 5-fold cross validation.



(b) Performance of SVMs in 4-class prediction task. SVM using linear kernel and overFEAT features gives the best performance. Similar results were obtained from binary prediction task.

Figure 3: Effects of parameter C and 4-class prediction BER performance of our constructed SVMs.

In above Figure.3(a), the x axis represents the parameter C of SVM while the y axis stands for BER. It can be observed that along with growth of C, the BER of our classifier firstly decrease step by step then keeps increasing. This result meets our expectation, as the parameter C in the SVM acts as a penalty factor – increasing C will increase the weight of misclassifications, which means the classification will be stricter. Hence, the results are similar to the penalty learning curve in logistic regression. We then set C to optimal values in all SVMs (C has different optimal values when using different featurs).

From Figure.3(b), it can also be observed that the SVMs using overFEAT feature give better BER performance than that using HOG feature, which implies overFEAT features contains more information than HOG features. Also, SVMs using linear kernel gives better BER performance than that using polynomial kernel (with polynomial order 2), hence, we employ SVMs with linear kernel as our final SVM classifier.

It should be noticed that we do not demonstrate the BER performance of SVMs using Gaussian kernel. Because we believe the simulation results are not stable and reliable. According to the validation matrix generated by SVMs with Gaussian kernel, all images will be assigned with some label. Also, we frequently experienced out of memory error in MATLAB. We believe that this may result from the extremely large computation which cannot be handled by our machines. Hence, we decided to drop Gaussian kernel in our model.

3.3 Neural Network Models

In this part, we introduce models based on neural networks. These neural network models are trained through many epochs (a single pass through the entire training set is an epoch). After finishing each epoch, we test the trained models by validation set and keep their accuracy. In order to avoid overfitting, only the models that have the highest prediction accuracy are used to predict label in the test set. All the accuracy values reported in this part are the highest validation accuracy in the training process.

3.3.1 Multi-Layer Perceptron (MLP)

Within Alex's work [2], to implement a 1000-category classifier on imageNet, he utilized 3 fully connected layers after convolutional layers and each fully connected layer contains 3072, 4096 and 1000 neurons respectively. In our task, the size of dataset and number of categories are much smaller, so our model contains fewer hidden layers with smaller size.

Alex's work in [2] also introduce a trick called 'dropout' to train a neural network. It randomly disable a neuron' output (reset to 0) with a probability p in a layer. This will reduce the co-adaptation between neurons and force the neurons to be trained relatively independently. In practice, this will relieve the over-fitting but lead to a slower converge. Figure.4(a) in the following page shows the learning curve of dropout model with different p. Larger p leads to a slower convergence.

Another trick is to use 'weight decay'. Similar to ridge regression, we add a punishing term, which equals to the norm of softmax weight multiplying a parameter λ, to the cost function. This mechanism force the softmax weight matrix to be sparser and improve the entire model's generalization to avoid overfitting.

The multi-layer preceptron model using HOG feature is the baseline in following Section 3.3.2, so in this part, we concentrate on overFEAT features. As mentioned in Section 2.2, we use ReLU as activation function. According Glorot's work in [3], we use normalized initialization $U(-\frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}})$ (n_{in} and n_{out} are number of neurons in the input and output layer) to initialize the weights of hidden layers and weights of softmax classifier are initalized 0.

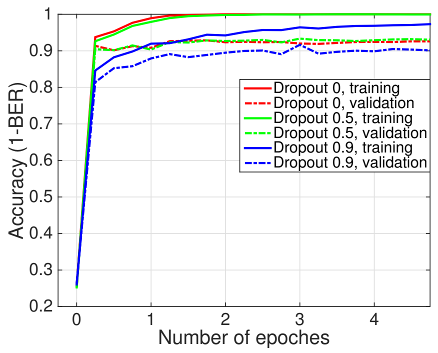
For binary classification, there are two methods. One is simply use the results of 4-category classifier and project it to 2 categories. The other one is directly train a binary classifier using 0-1 label. We have tried both methods and run each one for 5 times. The average balanced accuracy of the former model is a bit higher than the latter one (93.12% vs 92.47%). It is reasonable because the former model uses more information (fine-tuned labels other than binary ones) for training. For consistency, all the neural network models in section 3 use the 'projection' policy to generate binary prediction.

Table.1 below shows that both dropout and weight decay don't improve the performance. After taking a detailed look at the learning curve of this model (shown in Figure.4(a)), we did not see severe overfitting. That is why we did not see obvious improvement in dropout or weight decay

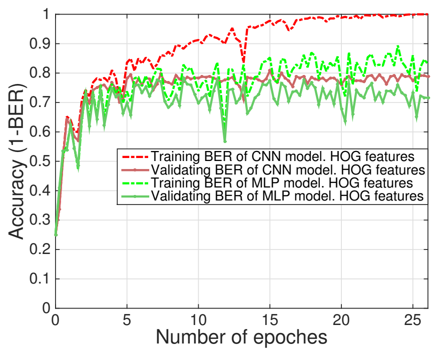
Model name	Hidden neurons	Remark	Accuracy2	Accuracy4	Run time(s)
MLP128	128	/	93.03%	92.93%	181
MLP256	256	/	93.05%	93.09%	372
MLP512	512	/	93.29%	93.19%	798
DropoutMLP	512	Dropout: $p = 0.5$	93.12%	93.08%	818
WDecayMLP	512	Wdecay: $\lambda = 10^{-3}$	92.79%	92.56%	804
MLP1024	1024	/	92.74%	92.87%	1703

Table 1: Performance of Different MLP Variants

model. In addition, the time spent on training a model is roughly linear to the number of neurons in the hidden layer. Actually, when the number of hidden neurons are too small, the model lacks capability to capture enough features. However, if the number of hidden neurons are too large, our training data will not be enough to train such a complex model (under-fitting), which will also hurt the performance. Besides the entries in the above table, we also tries model with more than one hidden layers, but it doesn't perform well and training process is very time-consuming. Besides lacking of enough training data, gradient vanishing in deeper model will also hurt the performance.



(a) Learning curve of Dropout MLP model. Larger dropout rate p leads to a slower convergence.



(b) Performance of MLP and CNN on HOG features.

Figure 4: Effects of dropout on MLP models and BER of CNN model using HOG features

3.3.2 Convolutional Neural Network

The convolutional neural network (CNN) was first proposed by LeCun in 1995 [4]. Recently it has achieved outstanding performance in computer vision tasks. It has one or more convolutional layers and in each layer, one instance is represented by a 3-dimensional tensor $X_l(i, j, k)$ of shape $[featuremaps, width, height]$. Let n_l and n_{l+1} be the number of feature maps in adjacent layers l and $l + 1$. There are $n_l * n_{l+1}$ filters W_{ij} whose shape is predefined manually between them. The input value of the following layer is the convolution of the last layer and the filters: $X_{l+1}(j) = \sum_{i=1}^{n_l} X_l(i) \otimes W_{ij}$.

As we can see, unlike fully-connected layer, the neurons in convolutional layers are only connected to the corresponding adjacent neurons of the lower layers. In addition, the number of trainable parameter is much smaller than fully-connected layers, which makes it possible for us to build a deeper network with acceptable number of parameters. These mechanisms make CNN powerful to extract local and hierarchical features. The activation function of neurons in CNN is usually ReLU, which can effectively solve the problem of gradient vanishing.

According to Alex's paper [2], what overFEAT features provide us is 1024-feature maps of size 6×6. It is already the output of a 5-layer CNN and the feature maps' size is relatively small, so in this part our models and results are based on HOG feature. This is reasonable, because neighboring areas tend to have similar gradient distribution pattern for CNN to capture. We consider each direction of gradient distribution as a feature map, so the given 5408-dimensional HOG feature set can be viewed as 32 13×13 feature maps.

We constructed a 2-layer CNN whose filters' shapes are 4×4 and 3×3 respectively, and each layer has 64 feature maps. After the first convolutional layer, we use a 2×2 max-pooling layer to reduce the size of feature map. After 2 convolutional layers, we use a softmax layer for classification.

The above Figure.4(b) shows the learning curves of MLP and CNN models based on HOG features. The MLP model has only one 512-neuron hidden layer. CNN model has much fewer trainable parameters than MLP, but the curves show that CNN model significantly outperforms MLP models both in respect of accuracy and stability.

3.3.3 Combined Neural Network Model

Previously, our models make use of only one type of features, either overFEAT or HOG. In this part, we will combine these two feature sets to train a more complex model. Because the best model for HOG features and overFEAT features is CNN and MLP respectively, we apply CNN to HOG and MLP to overFEAT to extract more abstract features and combine them as the input of softmax classifier (called HOG-CNN part and overFEAT-MLP part). The structure of this model is shown below. The hyper-parameter of MLP and CNN part is chosen from best models in section 3.3.1 and 3.3.2. For simplicity, we didn't draw max-pooling layer.

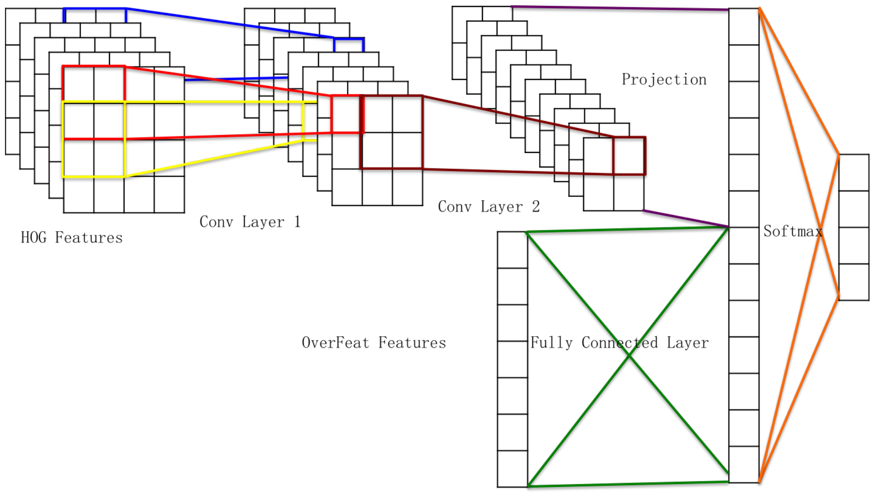


Figure 5: The Structure of Combined Neural Network Model

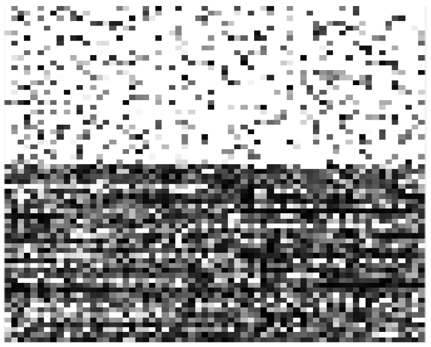
We run this model several times and the average accuracy is 93.02%, which is almost the same as pure MLP model based on overFEAT features. We believe it is because overFEAT-MLP part of this model converges much faster than the HOG-CNN part. overFEAT-MLP part are fine-tuned, the rest HOG-CNN part is still not well-trained. In this scenario, the weight of overFEAT-MLP features in the softmax classifier will be enlarged and those connected to HOG-CNN features will be weakened. The whole model will perform similarly as pure MLP models.

Figure.6(a) below shows the distribution of weight in the softmax layer. The pixel is lighter when the corresponding value's absolute value is larger. To be more clear, we reshaped the 1088×4 weight into 68×64, the first 32 rows are weights of overFEAT-MLP features and the rest are HOG-CNN features. It is clear that the upper part (overFEAT-MLP) are lighter than the bottom part (HOG-CNN). The mean weight of the two parts are 0.0222 and 0.0045 respectively. All these imply that overFEAT-MLP part dominates the model.

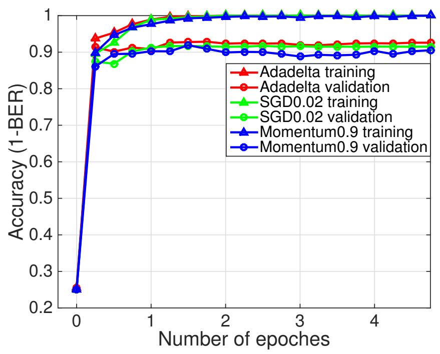
4 Optimization Methods

In this part, we use model MLP128 discussed in Section 3.3.1 to evaluate different optimization methods and examine their influence on the performance.

Given a function f and parameter x. The most naive method is stochastic gradient decent (SGD): the parameter moves in the opposite direction of gradient: $\Delta x = -\eta \frac{\partial f}{\partial x}$. The drawback of SGD is that



(a) Weight distribution of softmax layer in combined neural network model



(b) Performance of different optimization method used in model MLP128

Figure 6: Combined Neural Network Model and Optimization Method

it is easy to get stuck in local minimum. A revised version is momentum, this algorithm remembers the latest update and add it to present update by multiplying a parameter ρ call momentum: $\Delta x_t = \rho \Delta x_{t-1} - \eta \frac{\partial f}{\partial x}$. When Δx_t and Δx_{t-1} are of the same direction, the momentum accelerates the update step; while they are of the opposite direction, the algorithm tend to update in the former direction if x has been updated in this direction for many iterations. This mechanism helps x to go over the local minimum but makes the model more difficult to converge especially when ρ is big.

SGD and momentum are both first-order optimization method for they only take advantage of the first-order derivative $g = \frac{\partial f}{\partial x}$. Theoretically, second-order optimization methods are more accurate, e.g. Newton's method: $\Delta x = H^{-1} g$ where H is Hessian matrix. However, computing Hessian matrix is not computationally efficient. Adadelta [5] proposes a good method to estimate the Hessian matrix in linear time complexity. $\Delta x_t \simeq \frac{\frac{\partial f}{\partial x_t}}{\frac{\partial^2 f}{\partial x_t^2}}$, so $H^{-1} = \frac{1}{\frac{\partial^2 f}{\partial x_t^2}} \simeq \frac{\Delta x_t}{\frac{\partial f}{\partial x_t}} \simeq \frac{RMS(\Delta x)_{t-1}}{RMS(g)_t}$. Between the last approximately equal sign, we apply root mean square (RMS) to estimate Δx and $g = \frac{\partial f}{\partial x}$. For a variable X, $RMS(X)_t = \sqrt{E(X^2)_t + \epsilon}$ and $E(X^2)_t = \rho E(X^2)_{t-1} + (1-\rho) X_t^2$ where ϵ is a very small positive number and ρ is a predefined decay constant. One advantage of adadelta is that we do not need to set a model-sensitive learning rate manually and its convergence is relatively fast.

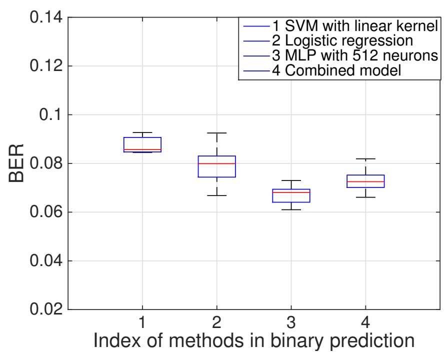
Figure.6(b) above shows the convergence of three methods in model MLP128. Model using adadelta update policy converges faster than other two methods and achieves the highest accuracy on validation set. On the contrary, momentum has the worst performance in respect of both convergence speed and accuracy. This is partly because its mechanism make it more easier to miss the minimum and thus slower the convergence.

5 Implementation

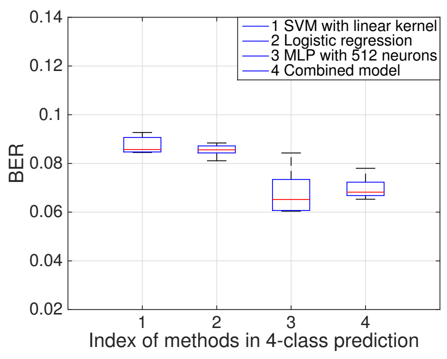
We implement models by Python and MATLAB. In our submit code, there are two subdirectory 'Python' and 'Matlab'. In each subdirectory, there are some models and each of them is in a folder.

Our MATLAB implementation uses packages provided by teacher. Our Python implementation uses some external machine learning package such as theano, sklearn, scipy. For neural network models, the code can be run much faster on GPU in a parallel manner. You can refer to 'readme.pdf' in top folder of submitted code to know more about how to run these codes.

All these codes are available on github (https://github.com/liuchen11/ImageClassification). Codes on github are more than what we discussed in this report, extra models like supervised version of self-organized map are included. You can clone this repository to download the full version of codes.



(a) Performance on Binary Prediction



(b) Performance on Multi-Class Prediction

Figure 7: Overview of Models' Performance

6 Conclusion

Figures 7(a) and 7(b) above show the overview performance of each model. It can be clearly observed that MLP model based on overFEAT feature performs best. So we employ it to generate predictions on test data.

In this project, we construct several classifiers to do image classification. The dataset is bigger than previous project and more challenging.

We gained a more insight understanding of SVM and neural networks after completing two classification tasks. Each of these models has many hyper-parameters that need to be set manually, such as kernel function in SVM and number of neurons in neural networks. Only after understanding their influence on models' complexity and capacity can we set them properly and get fine-tuned model.

Data analysis is still very important in this project. In addition to analyzing them directly, we need to know how to attain these features from images. Identifying properties of a given feature helps us employ suitable models for training.

The performance of some models are quite random, so we need to run each model several times. When evaluating models, we choose the model not only with high accuracy but also low variance.

7 Acknowledgement

We would like to thank Emti and all TAs, for offering their help and suggestions during this project. The codes of this project are implemented by group members independently. The report are written by Chen Liu and Mengjie Zhao, who are of the equal contribution. All rights of this report and published codes are reserved.

8 Reference

[1]. Dalal, Navneet, and Bill Triggs. "Histograms of oriented gradients for human detection." Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on. Vol. 1. IEEE, 2005.

[2]. Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

[3]. Glorot, Xavier, and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." International conference on artificial intelligence and statistics. 2010.

[4]. LeCun, Yann, and Yoshua Bengio. "Convolutional networks for images, speech, and time series." The handbook of brain theory and neural networks 3361.10 (1995).

[5]. Zeiler, Matthew D. "ADADELTA: An adaptive learning rate method." arXiv preprint arXiv:1212.5701 (2012).