Project-II by Group Shanghai

Xiaohui, Zeng
EPFL
xiaohui.zeng@epfl.ch

Ruofan, Zhou
EPFL
ruofan.zhou@epfl.ch

Abstract

In this report, we summarize our findings in the second project. We first visualized the features as to get more understand of them. And then we tried simple linear SVM and simple neural network on them. By analysing some of the "bad cases", we found that some of the label is wrong so that we applied data cleaning on our dataset and gave a sightly improvement on prediction. And then we tried different structures (i.e. different hyper parameters of the network) and different techniques such as dropout, activation functions on neural network and compared them to find out the best model.

1 project description

In this project, we are given 6000 images in size of 231x231 with labels in four classes including 'car', 'airplane', 'horse' and 'others'. Originally there are 966 airplane images(label 1), 1175 car images(label 2), 1519 horse images(label 3) and 2340 others image(label 4). Convolutional neural network (CNN) features in 36864 dimension and HOG[1] features in 5408 dimension are provided. We are also given CNN features and HOG features of test data without labels nor images. More specifically, the CNN features are extracted by the OverFeat[2] convolution network and they are exactly the output of the 5Th layer. Our task is to classify 11453 test data and predict their labels based on the given features and the training dataset.

2 Feature Visualization

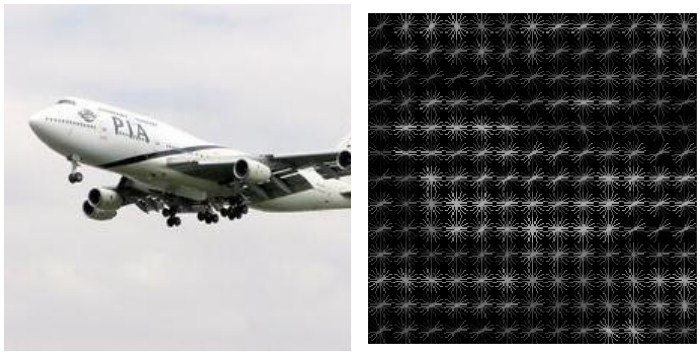
To gain more understanding of the features, we tried to visualize the features of the training data (i.e., we tried to plot the data based on its properties).

2.1 HOG feature

HOG features are extracted by computing an histogram of oriented gradients[1] in 13x13 image cell in equal size. We try to visualize the HOG feature of one of the airplane image as shown in 1(b). From the example we can see HOG could tell the boundary (or the contour) of the object to some extent.

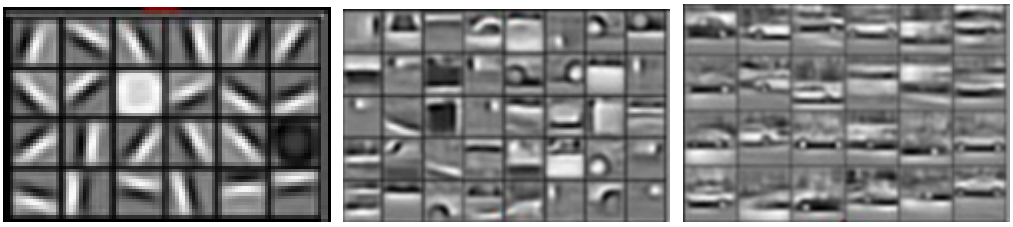
2.2 CNN feature

Though the weight of OverFeat[2] network is given, it's hard for us to visualize the feature due to the different coding library we use from the OverFeat. But from existing weight visualization of CNN (see example of "car" classification task from figure 2), we can conclude that the lower layer feature of CNN is "edge", middle layer feature of CNN is "part of the object", high layer feature of CNN is "different posture of the object". As our 36864-dimension CNN feature is obtained from 5th layer of the network, we can say that it should present "different posture of the object".



(a) image152 (b) HOG feature of image152

Figure 1: Visualization of HOG features



(a) weight visualization of CNN layer1 (b) weight visualization of CNN layer2 (c) weight visualization of CNN layer3

Figure 2: Visualization of CNN features (stretched from slides from Andrew Ng[3])

2.3 Feature selection

We finally choose CNN features for training and prediction since that the overall performance of the model (both SVM and Neural Network) build for CNN features is better than HOG features, which is reasonable since that CNN features are generally more informative and global than HOG features based on our knowledge that the extraction of HOG features loss quite a lot of information of the images comparing to convolution neural network features, and the references using HOG features are always knowing the bounding-box of the object while we don't have these data (and it's impossible for us to manually annotate these data). We also claim that our classification model is likely to perform better if the training are performed directly on the images so that the CNN features can be even more suitable for classification.

We tried CNN + HOG features on both SVM and Neural Network models, but there was just very little change (about $\pm 0.05\%$) in the prediction precision. Thus we can conclude that HOG feature couldn't help with the CNN feature, so we didn't use the combination of these two features in the later experiments.

3 Data Cleaning

When we tried to figure out the misclassified cases, we found that some of the given training data are actually mis-labeled (i.e., the label of them are wrong, see examples from figure3), and some of the data are confusing (i.e., we think that at least two of the labels could be assigned to the data, see examples from figure4). In this case, we relabeled on the subset (actually the subset we use is the "wrong answer" we get from the validation data, for we don't have time to look at and check the whole dataset) of the given training data and ignored those confusing data. Finally we corrected 54 pictures (about 0.9% of the whole dataset) and then fit the models on the cleaned dataset, which reduced about 1% of testing error in general for all the models.

(a) image36 (b) image581 (c) image5881

Figure 3: Some mis-labeled data. (a): A "horse" picture being labeled as "other"; (b): A "car" picture being labeled as "other"; (c): A "plane" picture being labeled as "other"

(a) image122 (b) image316 (c) image827

Figure 4: Some confusing data. (a): A picture containing a "car" but labeled as "other" (could be labeled as "car" or "other"); (b): An icon containing a horse-head-like pattern (could be labeled as "horse" or "other"); (c): A picture containing both horse and car (could be labeled as "horse" or "car")

We also observed that during the training process of Neural Network, the lowest validation error is obtained when the training error is around 0.05%, and as the training error is decreasing (when the training error is around 0.05%), the validation error is increasing. We guess the reason not only lies on "over-fitting", but also on these "wrong data". So we set a threshold for training, i.e., as training error is lower than 0.05% we will stop the training.

4 Model Selection

When it comes to the model selection, we held 20% of the data as testing (validation) set and 80% of the data as training set. The following model evaluation are performed on the same split of data for consistence. We evaluate our models with BER loss function while also looking into the model performance on different classes.

4.1 SVM

We tried to fit SVM model on our data. From Table 1, it is shown that SVM model with linear decision function outperforms SVM model with RBF model on our dataset, which indicate that the relation between the given CNN features and the labels of the images are more likely to be linear instead of non-linear.

Model	train error	test error
Linear SVM	-	0.0717
SVM with RBF kernel	-	0.0915
CNN fc512 with identity activation function	0	0.07128
CNN fc512 with ReLU	0	0.0752
CNN fc512 with dropout0.95	0.0143	**0.0625**
OverFeat last three layer	0.1682	0.2132

Table 1: CNN features model evaluation

4.2 Neural Network

Fully connected(fc) layer is widely used in most of the neural network currently as the last few layers for classification. It connects all the outputs of the previous layer, i,e every single neurons in the fully connected layer is computed by matrix multiplication of all the output of previous layer with some weights followed by a bias offset. By back propagation we can train the weights of each layer. Fc layer allow us to make use of all the given features for final classification and our experiment shows that it works well in our project.

We first tried to fit the neural network models with the same setting of the last three layers of the OverFeat network, i.e. an convolution layer taking the CNN features reshaped into 6 x 6 x 13 dimension followed by two large fully connected layer but it turns out that this model which gives test error of 0.2132, i.e. it did not give better result than our models. We believe that it is because the OverFeat model is originally designed for classification of 1000 classes of images which is far more complicated than our task. Therefore we designed an simpler fully connected layer for the project instead. And the result turns out that the simpler one is better than the complicate one.

We designed a simple neural network for 4 classes classification with one large hidden fully connected layer with 512 neurons taking all the CNN features as input and then followed by an smaller fully connected layer output 4 probabilities as the classes score. We denoted it as the fc512 model. The data is then labeled by finding the max probabilities among the four outputs, i.e class score. The model turns out to perform well and gives us the best test error of 0.0675.

We observed that the training error of the fc512 models reach 0 after several epoch while the test error remaining around 0.7. Considering the fact that the size of our training data is relatively small and limited, and as mentioned in previous section, there still remain some mis-labeled data in our training dataset, it is believed that the extremely low training error indicates over-fitting of our model, i.e., most of the noises in the training set are fitting by the complicated neural network model which will lead to classification error during testing.

To prevent overt-fitting and the effect by the mis-labeled data, we implement dropout layers after the fully connected layer. More specifically, we randomly remove some of the units out of the network together with their connection to other neurons, i.e., we tried to reduce the dependency of the model on the training data. It turns out that adding dropout did help with preventing over-fitting. The test error decreases while the train error increase a bit. Our experiment showed that dropout 95%, i.e. the dropout0.95 model, of the units randomly works better than other dropout models. We finally decided to use the model with 0.95 dropout according to the experiment shown in 5(a).

To further analysis the reason why dropout layer increase the model performance in general, we find that adding dropout layer can be viewed as averaging predictions of large amount of different models since that by randomly dropping the units, the network does not make identical prediction all the time[4]. In this way, the output probabilities for the correct class given by the network can be improved.

It also observed that the minimum of test (validation) error indeed is reached in some early epoch instead of reached at the end. Therefore we claimed that applying early stopping such as setting the stopping threshold for test error can be helpful during the fight with over-fitting.

We also tried another simpler neural network structure with 360 neurons and with dropout 0.95 in the fully connected layer to see whether the over-fitting situation can be improved. The experiment did give us higher training error with the test error remain about 0.07 which is almost the same.

Model	airplane	car	horse	others
CNN fc512 without activation function	**0.0732**	**0.0420**	0.1086	0.0571
CNN fc512 with ReLU	0.0773	0.0546	0.1054	0.0637
CNN fc512 with dropout0.5	0.0773	0.0462	0.9904	**0.0528**
OverFeat last three layer	0.0876	0.0462	**0.0959**	0.0945

Table 2: CNN features model evaluation for different class

groudtruth vs prediction	airplane	car	horse	others
airplane	181	1	1	11
car	0	227	1	10
horse	0	0	284	29
others	2	5	15	433

Table 3: confusion matrix for testing of CNN fc512 with dropout0.8 model, each columns is the prediction of test data while each row is the ground truth of the data

FC nodes number	train error	test error
512	**0**	**0.0752**
1024	0	0.0808
4096	0	0.0858
256	0	0.0875

Table 4: parameter tuning on FC layer

Several experiments of tuning the parameter of the number of nodes of the fully connected layer are also done, the result could be seen at 4. It's hard for us to tell the reason why 512 got the best results, so we just conclude from the experiment results to use 512 as the number of fully-connected nodes.

4.3 Model Evaluation

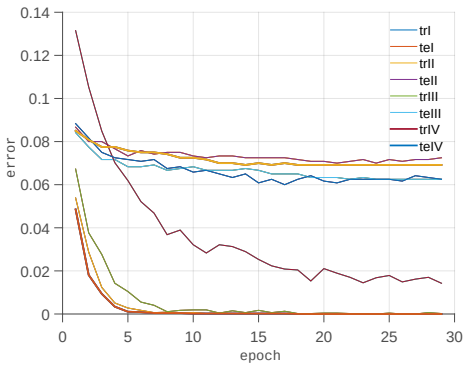
We noticed that the our the number of images in the training data are imbalanced, i.e. it contains many more images from one class then others. Typically there are more images with label 4(nearly 40% of the training data) than others. During the evaluation of our models, we also found that most of the error are produced by mis-classified image as label 4 as shown in 3. And most of the images labels wrong as label 4 comes from the 3rd class, i.e., some horse images are mis-labeled as others. It seems to be reasonable since that some of the horse images looks kind of similar with the images in fourth class. Another possible reason is that the model is trained on an unbalance dataset.

5 Experiment details

During the training of the neural network, we adapt the popular setting for some of the hyper parameter for the learning procedure due to the limitation of time and these setting did work in our case. We set the initial learning rate to be 0.001. The neural network are optimized by stochastic gradient descent on mini-batches of the training set and the size of batch is 100 with the momentum as 0.9, i.e. for each update of the parameters, it remain mostly the same as the original value but still keep updating. Our experiment result shows that the setting works well during the optimization and it is converging during the training.

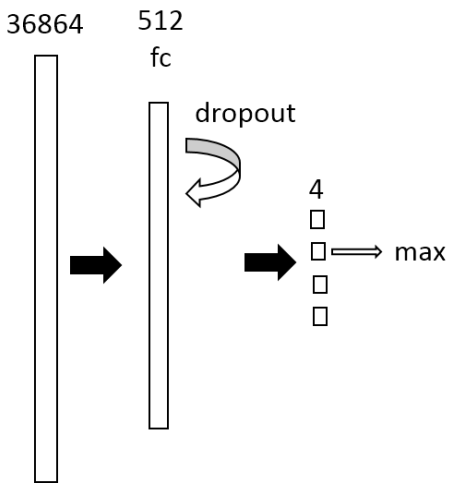
6 Implementation details

We used several machine learning libraries to help us to do the experiments, including Torch[5], Caffe[6], Theano[7], SKlearn[8], VL-feat[9], etc. The final code only uses Theano, so please make sure to have the library (including its dependencies such as numpy) installed before running the code.



(a)

Figure 5: train and test error for different models, I is fc512 with 0.6 dropout, II is fc512 with 0.8 dropout, III is fc512 with 0.95 dropout, adding dropout layer improve the performance in general



(a)

Figure 6: neural network structure

The file layers.py contains all definition and implementation of basic classes of neural network (including hidden layer, activation function, dropout setting, logistic layer, convolution layer etc, though some of them are used in training and experiments but not in prediction). And the file load_test.py is used to load test datas from mat file. The file cnn.py builds the network model, using it to predict the test data and then record the prediction of our own format. Finally you need to run change_format.m (in matlab) to changing the prediction into desired format.

We also include the training implementation cnn_train.py and load_data.py.

Acknowledgments

We would like to thank Emtiyaz Khan for making the dataset for this study. Thanks all the TAs for answering questions. And all the code, as well as this report, was written by Ruofan, ZHOU together with Xiaohui, ZENG.

References

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

[2] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, Yann LeCun. OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks.

[3] Andrew Ng. Unsupervised Feature Learning and Deep Learning Tutorial.

[4] Hinton, Geoffrey E., et al. "Improving neural networks by preventing co-adaptation of feature detectors." arXiv preprint arXiv:1207.0580 (2012).

[5] Amazon EC2 image with torch

[6] Jia, Yangqing and Shelhamer, Evan and Donahue, Jeff and Karayev, Sergey and Long, Jonathan and Girshick, Ross and Guadarrama, Sergio and Darrell, Trevor. Caffe: Convolutional Architecture for Fast Feature Embedding. arXiv preprint arXiv:1408.5093 (2014).

[7] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. Goodfellow, A. Bergeron, N. Bouchard, D. Warde-Farley and Y. Bengio. Theano: new features and speed improvements. NIPS 2012 deep learning workshop.

[8] Pedregosa, F. and Varoquaux, G. and Gramfort, A. and Michel, V. and Thirion, B. and Grisel, O. and Blondel, M. and Prettenhofer, P. and Weiss, R. and Dubourg, V. and Vanderplas, J. and Passos, A. and Cournapeau, D. and Brucher, M. and Perrot, M. and Duchesnay, E. Scikit-learn: Machine Learning in python.

[9] A. Vedaldi and B. Fulkerson. VLFeat: An Open and Portable Library.