Various Approaches to Image Recognition

submitted for fulfilling the project requirements of the course

CS 289A – Introduction to Machine Learning

Ninh Hai DO
Department of Mechanical Engineering
University of California at Berkeley
CA, USA 94720
ninhdo@berkeley.edu

***Abstract* — this paper investigates the image recognition problem using various classification method learned from the course CS 289A. Those methods categorized into two groups: nonlinear methods includes convolutional neural networks and k-nearest neighbors; and linear methods encompasses ridge regression, linear discriminant analysis (LDA), quadratic discriminant analysis (QDA) and support vector machine (SVM). After all, I realize that the nonlinear method, typically convolutional neural network, works best for the project whose nature involves data collected from real world.**

Keywords—Convulutional, Neural Network, Image, Recognition

I. BACKGROUND

The company AxleHire developed a mobile application connecting cargo drivers and clients. Drivers have to take a picture of cargo boxes against the background of delivery locations as a proof of completing delivery. Some drivers are so reckless that they shoot arbitrary pictures as they know nobody checks them. They are somewhat correct: the number of pictures went up to hundreds thousands after two years, which deterred anybody who ever thought of checking them manually.

During last Summer I worked as a volunteer intern at the company. At the end of my internship, I was exposed to the problem and started working on it from September, with the company's permission to access its database. I developed a convolutional neural network model to recognize if there is a package in a given image. That is, the model can classify images into two categories "package" and "no package" after being trained on a huge sources of image data. Next step, the model is able to localize a "package" if an image has it.

Since this is a real-world project, there are several challenges emerging during its implementation. I will discuss most of them in the following parts. Within the scope of a course project, and for the project is still in progress, I just present its first part that is the implementation of convolutional neural network to classify images into two categories.

II. DATA AND PROCESSING DATA

A. *Data Overview*

The data set is a pool of RGB images. Its whole data size is ~ 80.35 Gigabytes including 407,936 images. To reduce the workload, I randomly sample the subset of 5.35 Gigabytes consisting of 28,494 images. The data set is divided into two subsets with ratio 5:1. The first one is the training data set, the second is the validating data set. After the model is trained, it is tested on a new randomly selected sample.

The data is raw. It does mean that the images have not been labelled. For the purpose of training and validation, I had to manually classify the image subsets and label them properly. Besides, there are several problems involving the data set such as:

- It has noises. Some images are totally black or white.
- The images are shot at different angles. Some are horizontal, vertical or upside down.
- The images have different resolution, since drivers have different phones and they shoot pictures with different qualities.
- The images are shot at different times: morning, afternoon, evening. Etc.

To give a better idea about the quality of the data set, some images are presented as below:

Figure 1. Images with packages

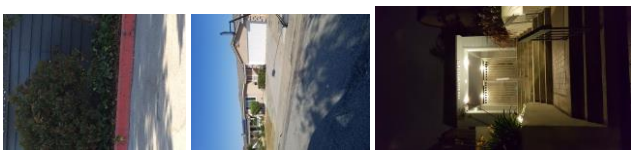

Figure 2. Images without packages

B. *Data processing*

For the nature of the project, when the model is deployed and integrated into the mobile app, it must be able to recognize packages in low-quality images, in other words, the model will not always work with "nice" images. Thus, I decided not to clean data. By training a model on an unclean dataset, the accuracy may be lower but the model becomes more stable and this reduces the chance the model give surprisingly low accuracy when it works in real world, because the real world consists of images of all kind of qualities and noises and the model has been trained on such kind of images beforehand.

I process the data set as following:

- Make the data set more homogeneous: convert all the images into square image of size 32 x 30 with format png.
- Make the data set more uniform. As mentioned above, the images are of different qualities and orientation. Before the images are fed into training, they are randomly blurred, rotated and distorted to some extent.

Some images after processed are presented as below:

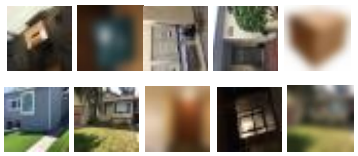

Figure 3. Images after being processed

where the first row is the images with packages, among which the second and the fifth is blurred and the third is rotated 90 degree, and the second row is the images without packages.

III. Convolutional Neural Network Model

A. *Architecture*

The convolutional neural network model has the following architecture:

- Convolutional layer 1: Add bias and apply ReLU activation.
- Apply max pooling.
- Normalize local response.
- Convolutional Layer 2: Add bias and apply ReLU activation.
- Apply max pooling.
- Normalize local response.
- Fully connected layer with ReLU
- Fully connected layer with ReLU
- Softmax layer

The CNN architecture is presented in the following diagram.

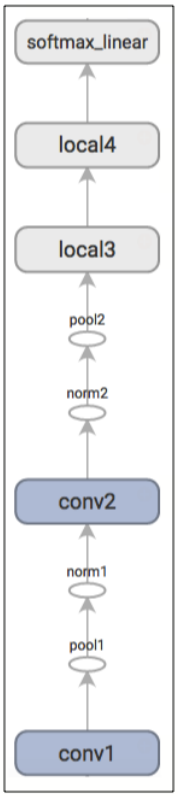

Figure 4. The CNN architecture (a courtesy of TensorFlow)

B. *Training and validation*

The training is conducted on the data set of 23,000 images over 10,000 batches with the log frequency of 10. It take around 5 hours to run on a quad core laptop 2.8 GHz. Plotting the training loss over number of batches gives us the below diagram:

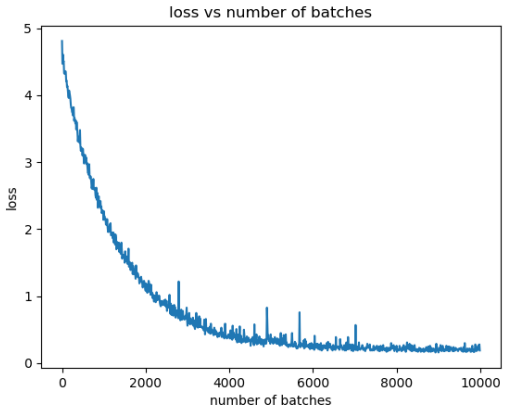

Figure 5. Loss versus number of batches for training data set of 32x32 images.

As we can see in the above diagram, the loss drops consistently over 7,000 batches. After that, the loss tends to reach a plateau, which indicates that the constraint of the model has been reached.

Subsequently, I evaluate the trained model on the separate validating data set of 5494 images (which is around 1/5 of the training data set size), the accuracy is 92.12%.

For tuning hyper-parameters, there are a lot of features that we can take into account. Due to the time constraint for training the model, I just pay attention to two parameters: image size and number of batches. The summary of tuning parameters is presented in the following diagram:

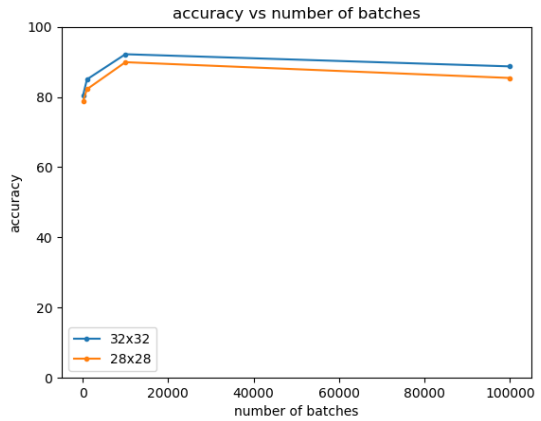

Figure 6. Accuracy versus number of batches for training data set of 28x28 and 32x32 images over 1000, 10000, 100000 batches.

From the above diagram, the best hyper-parameters are size 32 x 32 of image with 10,000 batches of training. Of course, this tuning is quite bias due to a small range of features. But it works conceptually. If the size of image increase, the result is expected to be better.

C. *Coding Structure and Running Instructions*

The project is coded in Python 2.7. The dependency packages are indicated in requirements.txt. The coding has multiple components. To run it, follow the steps:

1. *Train the model, type:*

   $ python cpu_train.py

   The MAX_STEPS in cpu_train.py indicates the number of training batches. It can be changed. Typically, training the model over 10,000 batches takes ~5 hours, over 1,000 batches takes more than 30 mins.

2. *Evaluate the model, type:*

   $ python eval.py

   This will indicate the accuracy percentage. Typically, the 10,000-batch trained model has the accuracy of 92.12%, the 100,000-batch trained model is 88.7% accurate.

3. *Run the model, type:*

   $ python main <path to image file/folder>

   The output tell whether there is box or no box in the image(s).

IV. K-NEAREST NEIGHBORS MODEL

A. *Model description*

The k-nearest neighbor model follows the same method taught in the class. That is, given the training data set, I investigate the label of k nearest neighbors to the new data point and assign the label of major points to it. The implementation of this model is pretty simple using library "sklearn". The data, however, must be flattened first. Subsequently, the training data are fitted into the model, and the prediction of evaluation data is based on the trained model. Surprisingly, the result is not bad.

B. *Coding Structure and Running Instructions*

From the coding/k-NN, type:

$ python train_nn.py

C. *Result*

The accuracy diagram is presented as below.

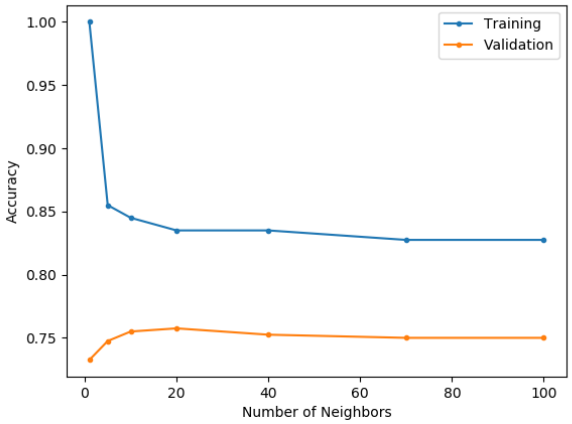

Figure 7. Accuracy versus number of neighbors.

As we can see in the above diagram, the performance is improved when we increase the number of neighbors k, then it reaches a plateau. Typically, the best parameter for this method is k = 20 (or around 20).

V. LINEAR MODELS

A. *Description*

In this section, I will implement a wide variety of linear models presented in the class. Typically, those are ridge regression, Linear Discriminant Analysis (LDA), Quadratic

Discriminant Analysis (QDA) and Support Vector Machine (SVM). Again, I take advantage of library sklearn and reuse the confusion matrix coding to present the accuracy. Unfortunately, the result is not good compared to the similar problem presented in homework 10. This is understandable and will be discussed at the end.

B. *Ridge Regression Model*

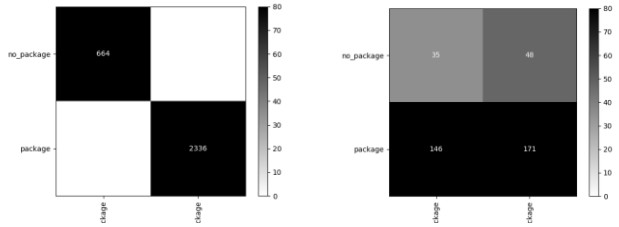

Figure 8. Accuracy versus number of neighbors for ridge regression model.

C. *Linear Discriminant Analysis (LDA)*

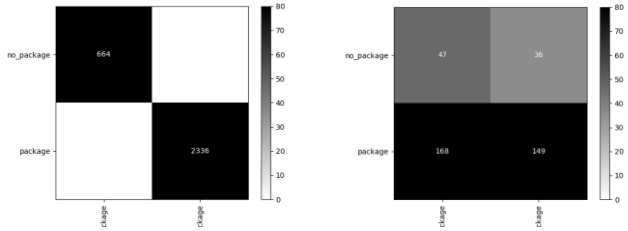

Figure 9. Accuracy versus number of neighbors for LDA model.

D. *Quadratic Discriminant Analysis (QDA)*

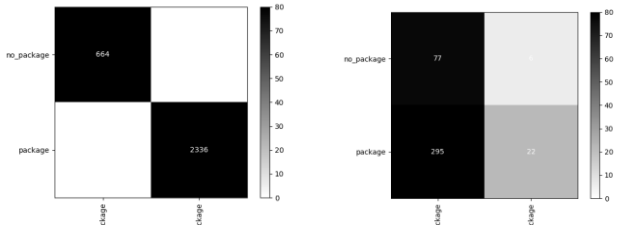

Figure 10. Accuracy versus number of neighbors for QDA model.

E. *Support Vector Machine (SVM)*

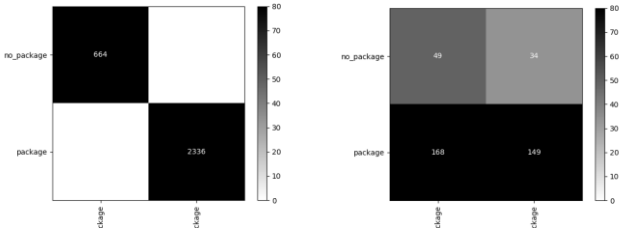

Figure 11. Accuracy versus number of neighbors for SVM model.

F. *Coding Structure and Running Instructions*

From the coding/Linear_Models directory, type:

$ python linear_classification.py

G. *Discussion*

As we can see in all of the diagrams of the linear models, the training accuracy is very good but the validation accuracy is not much better than the base line. The reason is the difference of images with package and without package is very subtle, and the images are shot in random way. This increase the nonlinearity of the problem and the linear classifier cannot work well.

VI. Conclusion

To the end, we can see the convolutional neural network outperforms the other methods. It is typically good for image classification problem. Besides, the k-nearest neighbor method is also work well. All linear models are bad due to the nonlinearity of the problem.

References

[1] CIFAR10, TensorFlow

[2] Sahai, A. Yu, S., Lecture Notes, Homeworks and Coding