IMPROVEMENT OF DEEP LEARNING MODELS ON CLASSIFICATION TASKS USING HAAR TRANSFORM AND MODEL ENSEMBLE

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Bachelor's thesis

Valkeakoski, Automation Engineering

Spring 2017

Tung Son Nguyen

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

ABSTRACT

Automation Engineering
HAMK Valkeakoski

Author	Tung Son Nguyen	**Year** 2017
Subject	Improvement of deep learning models on classification task using Haar transform and model ensemble.	
Supervisors	Raine Lehto (Raine.Lehto@hamk.fi – HAMK University of Applied Sciences) Joni Kämäräinen (joni.kamarainen@tut.fi – Tampere University of Technology)	

ABSTRACT

Machine learning have an enormous impact on Computer Vision. This thesis investigates how to improve efficiency of a Machine learning technique called deep learning on classification tasks using Haar transform and model ensemble. Haar transform can be used to reduce the input size of image data to train more models and model ensemble is known to boost performance using multiple models instead of one.

Experimental results showed that Adaboost and stacking work as expected as they boosted the accuracies by 2-3% and approximately 1%. However, the other two methods, averaging and geometric mean, did not boost but scored between the best individual model and the worst. This thesis also suggests future work onto Adaboost and stacking.

Keywords machine learning, deep learning, computer vision, haar transform, model ensemble

Pages 27 pages including appendices 6 pages

CONTENTS

1 MOTIVATION

Machine learning is advancing very fast nowadays with the support of powerful hardware and parallel programming. From self-driving cars to face recognition and lung cancer detection, machine learning is there to help to solve difficult tasks. Some of them even seem impossible. With a belief in the enormous potential of this field, scientists and engineers are racing to develop new algorithms and techniques which can beat the current best. One of these recent techniques are convolutional neural networks or deep learning in general. First proposed by Yann LeCun (LeCun et al., 1989), deep learning made its first success in classifying hand written digits. Over the time, deep learning has become a very effective method with good generalization, reasonable training time thanks to GPU technology and very little feature engineering. These features make deep learning an excellent candidate for image-related work. However, can we do better? This thesis examines a method of using Haar wavelet transform and model ensemble to boost the performance of deep learning models.

Model ensemble has been a well-known method to boost the accuracy of a specific machine learning algorithm just by combining different models trained on different dataset. But is there any other way of producing different training sets rather than splitting the original one?

Haar wavelet transform is a possible answer. Haar wavelet transform is an image processing technique that is often used in image compression. The basic idea is that Haar transform can reduce the dimensions of an image by a power of 2, therefore reduce the workload or memory of transferring or storing the original image. The smaller image can always be transformed back to original dimensions using inverse Haar transform. With the use of Haar transform, images can be reduced to different smaller dimensions which means that different training sets can be produced.

2 INTRODUCTION TO MACHINE LEARNING AND DEEP LEARNING

2.1 Machine learning

Machine learning is a subfield of Artificial Intelligence that “gives computers the ability to learn without being explicitly programmed” (Samuel, 1959). Machine learning has an enormous impact on many different areas, this thesis however discusses only one application: a classification task in computer vision.

In classification task in computer vision, a set of images is collected beforehand. The images are coloured or grayscale images of the same size and each one of them is labelled correctly as the name of the object appearing in the image. For example, an image containing an airplane will be labelled “airplane”. The labelled images are then given to a computer and throughout an algorithm, the computer will build a model to extract knowledge from these images. This process is called **training**. After this, the model will be given images which it has not seen before to classify these new images into labels (or **classes**) which are known during training. This process is called **predicting**.

Labelled images are often divided into three subsets: **training set**, **validation set**, **testing set** by 80:10:10. The training set and the validation set are involved in training process to diagnose what is currently wrong with the model while testing set is used in predicting process along with some accuracy metrics to see how well the model is classifying.

For simplicity purpose, collecting images was skipped in this thesis and a public data set CIFAR-10 (Krizhevsky, 2009) was used for the experiment. This data set contains 60000 coloured images of size 32x32 and each one of them is labelled with the name of a single object they contain: “airplane”, “automobile”, “bird”, “cat”, “deer”, “dog”, “frog”, “horse”, “ship” and “truck”.

2.2 Deep learning

Deep learning is a machine learning technique that is used during the training process to extract knowledge from the data. It has a unique nature that it requires very little or no input pre-processing to learn very well from training data unlike other machine learning techniques. Also, as the GPU hardware is being improved and becoming better and more inexpensive, deep learning is favoured more and more since training time is much shorter than before. All of this makes deep learning a perfect solution to computer vision or speech recognition problems.

2.2.1 Neural network

A neural network (M. W. Gardner, S. R. Dorling, 1997) is the core concept of deep learning models. Inspired by human brains, neural network consists of multiple layers which are fully connected to each other. Each layer also contains multiple nodes or neurons. There are three layer types in a neural network:

- Input layer: where data is feed into the network.
- Hidden layer: where output from input layer is processed, or activated. A neural network can have more than one hidden layer.
- Output layer: where output from hidden layers is forced into probabilities of class labels.

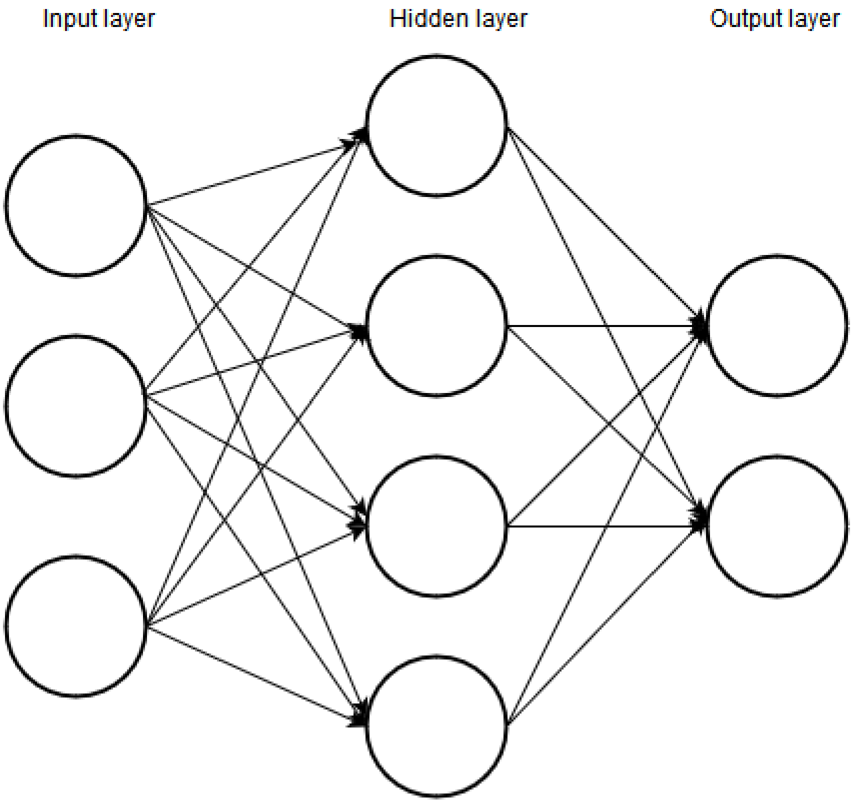

Figure 1: A neural network that has 1 input layer, 1 hidden layer and 1 output layer.

2.2.2 Convolutional neural network

A convolutional neural network (CNN) (LeCun et al., 1998) is very similar to a neural network in the sense that they both share the same structure and idea. The only difference is that CNNs have more layers and several types of layers. This section quickly introduces some of the most common layer in a convolutional neural network besides input and output layers.

- Convolutional (CONV) layer is the core layer of a CNN (LeCun et al., 1998). It filters and captures only the local regions in the input. Figure 2 illustrates how convolutional layer works. Suppose we have an input of 7x7 (bottom square) fed into a CONV layer. This layer will then look at each smaller region of the input and map them into an 5x5 yet smaller output (top square).

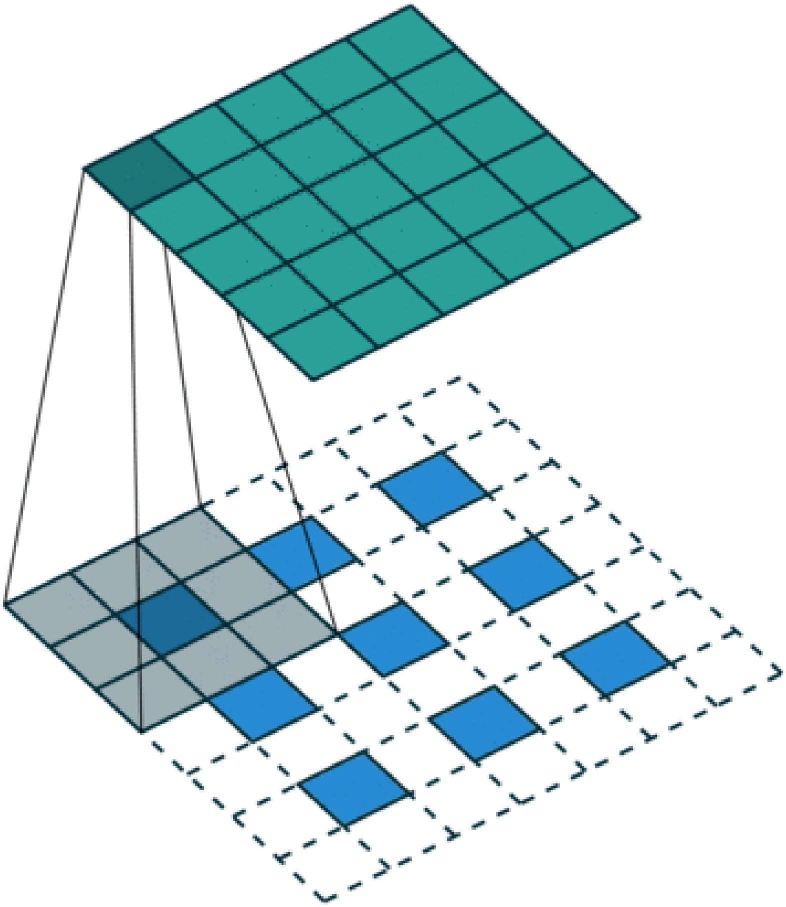

Figure 2: An example of Convolutional layer. (understanding-convolutional-neural-networks-for-nlp, 2015)

- RELU layer applies activation function and leaves the dimension of the input unchanged (LeCun et al., 1998).
- Max pooling (POOL) layer shrinks the input dimension by a factor (LeCun et al., 1998).
- Besides the most common and important layers above, there are some other types such as DROPOUT, FLATTEN or DENSE etc. Some of them are used to either change input dimension or prevent overfitting. These layers will be explained briefly as they appear later in this thesis.

A complete architecture of a CNN often contains multiple combinations of three above layers before forwards everything into some fully connected layers to produce probabilities. CONV-RELU-POOL combinations help CNNs generalize input data very well without overfitting since they only compute smaller areas instead of every single number in a matrix input. Below is an illustrated architecture of a typical CNN:

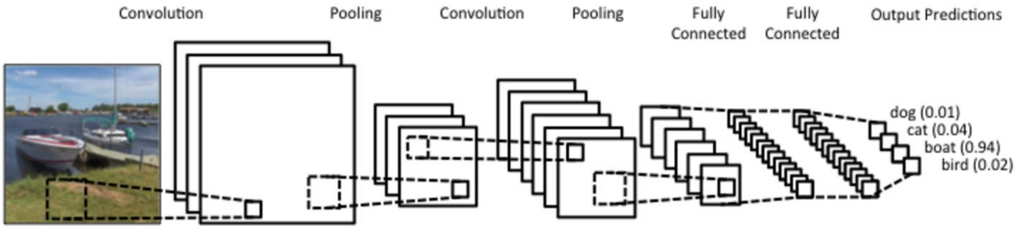

Figure 3: A typical CNN with image input data, 2 combinations of CONV-RELU-POOL, 2 fully connected layers and produces probabilities for 4 different classes. (CONV layer, 2016)

3 DEEP LEARNING MODELS

3.1 Architecture

The architecture of the deep neural network used in this experiment is very simple but powerful. There are two main groups of layers in the model: the deep layers and the fully-connected layers. Deep layer groups consist of two combinations of CONV-RELU with MAX-POOLING as the last layer. Fully-connected layer group is more like a regular neural network as it contains two layers which are connected to each other. The structure of deep neural network is illustrated in the following:

$$Deep\ layers \left\{ \begin{array}{c} conv \\ relu \\ conv \\ relu \\ pooling \\ dropout \end{array} \right.$$

$$Deep\ layers \left\{ \begin{array}{c} conv \\ relu \\ conv \\ relu \\ pooling \\ dropout \end{array} \right.$$

$$Fully\ connected\ layers \left\{ \begin{array}{c} flatten \\ dense \\ relu \\ dropout \\ dense \\ softmax \end{array} \right.$$

The basic flow when an image is fed into the model is simple. First, the image goes into two CONV-RELU so that important regions in the image are captured. Then, MAX POOLING will shrink down these regions a little bit in size. Next, DROPOUT sets some random weights to zero to prevent the model to overfit (Srivastava et al., 2014). Similarly, there is a second deep-layer group to continue doing the same thing. Finally, fully-connected layers come (Hinton et al., 2012). The input at this point is three dimensional, so a FLATTEN layer is needed to reshape it into one dimension, i.e., a row vector. This row vector is now fed into two DENSE layers which are nothing but regular hidden layers. RELU and DROPOUT come between these DENSE layers for input activation and preventing overfitting. SOFTMAX is the last layer, it forces the output of the models into probabilities.

3.2 **Training and testing**

3.2.1 Training algorithm

In this project, deep learning models were trained by Stochastic Gradient Descent (SGD) algorithm (Bottou, 2010). The idea of this algorithm is simple and is demonstrated in the following.
First, SGD starts with the following objective function:

$$Q_{(w)} = \frac{1}{n}\sum\nolimits_{i=1}^{n} Q_{i(w)}$$

$$with\ w\ is\ the\ parameters\ being\ estimated$$

- This function measures the loss or how bad the model is fitting the training samples with parameter w, so we want to find w which minimizes this loss. SGD starts with an initial random guess of w. It then iterates through n training samples and for each sample, it computes the gradient of the loss at this sample with respect to w; i.e.; $\frac{\partial Q_i}{w}$. According to Robbins-Siegmund theorem (H. Robbins, D. Siegmund, 1971), convergence or a global minimum of the loss is almost sure to be obtained. Hence, to converge, SGD updates w by:

$$\mathrm{w} = w - n\frac{\partial Q_i}{w}$$

$$where\ n\ is\ the\ learning\ rate\ or\ step\ size$$

To sum up, a complete pseudocode of SGD algorithm looks like this:

```
• Choose an initial vector of parameters w and learning rate n
• Repeat until a global minimum is obtained:
    o Shuffle training samples uniformly
    o For $i = 1, 2, 3, \dots, n$ do:
        ▪ $w = w - n\frac{\partial Q_i}{w}$
```

3.2.2 Testing procedure

First, the training set is divided into two parts. One part is used for training, the other one is used for validating. Now, for each epoch, the model is trained on the training set, then the probabilities on the validation set are predicted and the accuracy is reported. By doing this, one can understand how the model gets better at classifying after each epoch.

Finally, after the model have been trained for many epochs, it is used to predict on the test set. This test set is different from the training and validation set and is provided by Kaggle (CIFAR-10 - Object Recognition in Images, n.d.). At this point, predictions are uploaded to Kaggle and they report the accuracy back. This is now the **official accuracy** of this model.
Table 1 reports characteristics of four models which are trained with the same algorithm for 100 epochs:

Name	Train loss	Train acc	Val loss	Val acc	Train time	Final acc
32x32	0.5978	0.7923	0.5199	0.8224	2439.06	0.8241
16x16 haar	1.1957	0.5858	0.9376	0.6768	1063.56	0.6706
16x16 diff	1.0403	0.6342	0.9999	0.6570	1092.88	0.6574
8x8 haar	1.57	0.4479	1.4224	0.51	839.22	0.5157

Table 1 - Accuracies of deep learning models on different datasets

- Models are named based on the dataset they were trained on.
- Train loss and Val loss are the losses computed by the SGD loss function in Section 3.2.1 on the training set and validation set, respectively.
- Train acc and Val acc are the accuracies of the predictions on the training set and validation set, respectively which are computed by dividing the number of correctly classified samples with the total samples.
- Train time is the actual time in seconds that each model spent on training.
- Final acc is the official accuracy of each model which is reported by Kaggle.

4 HAAR TRANSFORM

4.1 Basics

Haar wavelet transform is a dimension reduction technique which is often used for time series data and images. The Haar transform can be expressed in the following matrix form (Rafael C. Gonzalez, Richard E. Woods):

$$T = HFH^T$$

where F is an $N \times N$ image matrix, H is an $N \times N$ Haar transformation matrix and T is the resulting $N \times N$ transform. To generate matrix H, Haar basics function is used. Some examples of matrix H are:

$$H_2 = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \text{ (2×2 Haar matrix)}$$

$$H_4 = \frac{1}{\sqrt{4}}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ \sqrt{2} & -\sqrt{2} & 0 & 0 \\ 0 & 0 & \sqrt{2} & -\sqrt{2} \end{bmatrix} \text{ (4×4 Haar matrix)}$$

Details on how to generate these Haar matrices is more complicated hence not presented here.

Figure 4 is an example of how Haar transform works on images. Using MATLAB command *haart2*, two different images could be produced from a 32x32 image of a frog. The resulted images are smaller and less details are displayed, but still shows the shape of the frog.

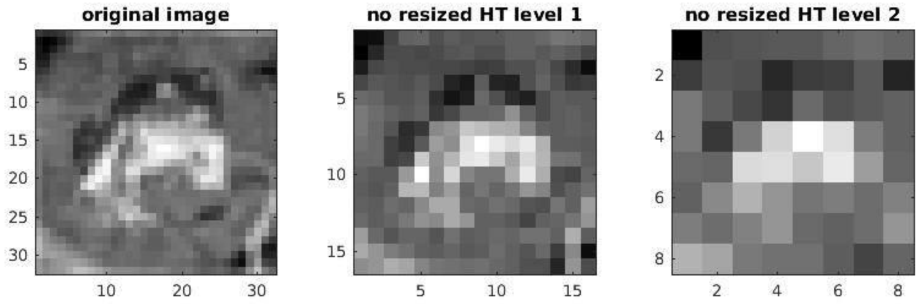

Figure 4: An example of using Haar transform compress a 32x32 image into smaller dimensions: 16x16 and 8x8

4.2 Illustrations

Figure 5 plots five different images of “car”, “deer”, “truck”, “truck” and “horse” before and after Haar transformation.

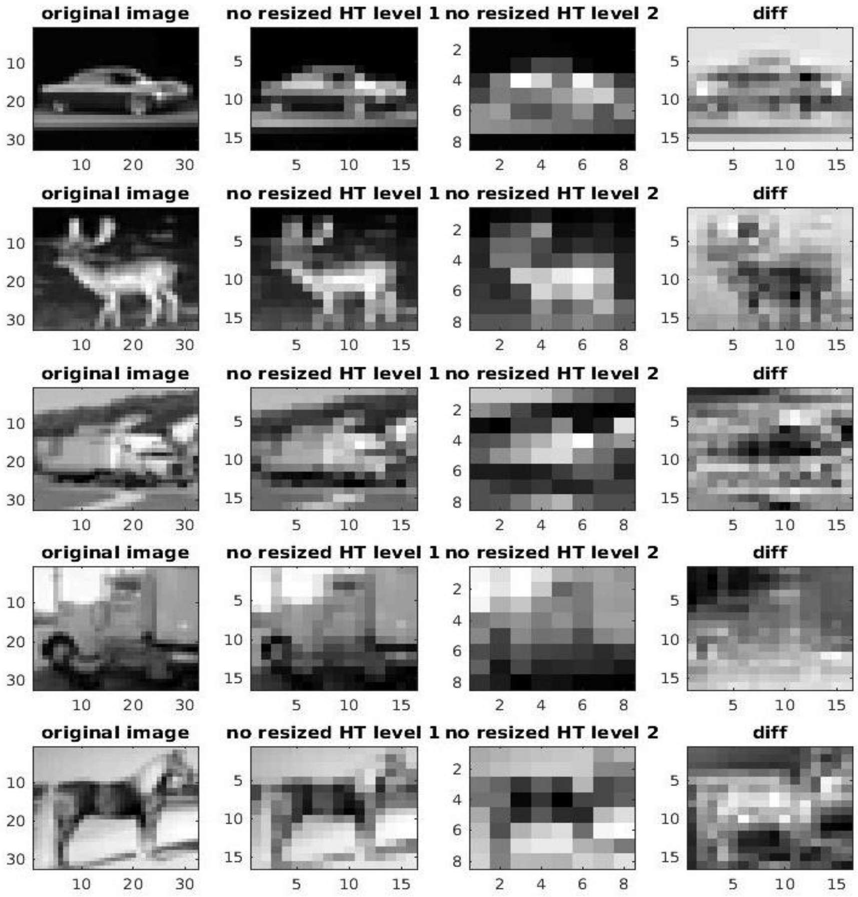

Figure 5: Some other examples of Haar transform on different images from the training set

4.3 Producing images using Haar transform

Since the CIFAR-10 dataset is very big (50000 images), how to perform Haar transform efficiently and program this task in MATLAB was a problem. I shall begin by analysing the original solution, then pointing out two improvements that will speed up the entire process a lot.

4.3.1 Inefficient MATLAB program for performing Haar transform

```
% Init an empty vector
haarlvl1 = []

for j = 1: 50000
    % Read j-th image
    img = rgb2gray(imread(sprintf('train/%d.png', j)));

    % Perform haar transform
    h16 = haar_function(img);

    % Flatten the haar image
    h16 = reshape(h16, [1, 16*16]);

    % Push it to vector
    haarlvl1 = [ haarlvl1; h16];
end
```

First, an empty vector is initialized. Then the program looped through 50000 images, for each image read, Haar transform is performed on it and finally the image is pushed into my vector. This program worked correctly, however, there were two problems concerning its performance:

- Initialize an empty vector was not a clever idea, especially when the size of the vector was known after reading. The reason for this was when a new image was pushed into my vector, the program had to copy the whole thing before. Therefore, the asymptotic complexity is $O(n^3)$ where n is the number of images (50000 in this case).
- Parallel for loop could be used to speed up the loop.

4.3.2 Better MATLAB program for performing Haar transform

Using these above analysis, I could program this task in a much more efficient way:

```
% Init an zeros vector with the size (50000, 256)
haarlvl1 = zeros(50000, 16*16);

parfor j = 1: 50000
    % Read j-th image
    img = rgb2gray(imread(sprintf('train/%d.png', j)));

    % Perform haar transform
    h16 = haar_function(img);

    % Flatten the haar image
    h16 = reshape(h16, [1, 16*16]);

    % Push it to vector
    haarlvl1(j, :) = h16;
end
```

- Now because I knew the size of my vector, I initialized it with proper size beforehand. My program didn't have to copy the whole thing when it added new image, so the performance is $O(n)$ and it was $n^2 = 50000^2$ times faster. A huge improve!
- I could also use "parfor" which stands for "parallel for" instead of "for", so MATLAB knew I wanted my CPU to work in multiple cores instead of a single core.

4.3.3 Outcome

The outcome of this program is that we now have four different datasets. The first one is the original dataset which contains 32x32 images. The next two is the haar level 1 and haar level 2 dataset which contains 16x16 and 8x8 images transformed from the first one. The last one is the difference images between the second and the third which are 16x16 images. Readers can consider section 4.2 to get an idea of these datasets.

5 MODEL ENSEMBLING

We now at this point have multiple datasets. We also see that single deep learning model is doing very well on classification task (more than 80% classified correctly). Is it the limit? Can multiple models do better than a single model? This is where model ensemble comes in. By combining two or more models, a higher accuracy is expected to be achieves. But how to combine and which models to combine?

5.1 Boosting

Adaboost (Zhu et al., 2006) is an ensemble algorithm known for combining several models by invoking them many times on a training set. The idea of this algorithm is very simple. First, we weight every sample of the training set by a constant and train a model on this set. Next, we increase the weights of the misclassified samples and decrease the weights of the correctly classified samples. This process will happen multiple times and after this, we will have a pool of models with different weights for each model. However, there are a few things to notice:

- The models to combine should be “weak”. That means their prediction capabilities are limited. Otherwise, this method will only lead to overfitting.
- Deep learning models are clearly not “weak”. But if Haar transform is applied to the data so that the information that the model receives is reduced, a “weak” model will be trained. Experiments have confirmed this. On level two Haar transformed images, deep learning model classified ~50% correctly, in contrast of ~80% when trained on the original images.
- The original Adaboost algorithm trains the models with a set of sample weights. This, however, does not work with deep neural networks without adapting suitable learning rate (Holger Schwenk, Yoshua Bengio, 2000). Since adapting suitable learning rate can be a heavy and slow task, a different algorithm has been chosen. Instead of training the model with sample weights, this second algorithm resamples the training set so that the probability of misclassified samples is high.

This section first introduces and explains pseudo-code of the Adaboost algorithm by words and computer codes. Next, results of experiments on CIFAR-10 dataset is presented.

5.1.1 Multiclass Adaboost algorithm

The algorithm is invented by (Holger Schwenk, Yoshua Bengio, 2000). However, the algorithm presented below has been slightly modified so that it is easier to understand and program.

```
Input: sequence of N examples $(x_1, y_1), \dots, (x_N, y_N)$ with labels $y_i \in \{1, \dots, K\}$
Initialize: sample weights $D_1(i) = [D_1(i,1), D_1(i,2), D_1(i,3), \dots, D_1(i,K)]$
with $\begin{cases} D_1(i,j) = \frac{1}{N(K-1)}\ if\ y_j \neq y_i \\ D_1(i,j) = 0\ otherwise \end{cases}$
Repeat:
    1. If this is the first estimator, train the estimator on all the training set.
       Else, resample the training set with respect to distribution $D_t$
    2. Obtain hypothesis $h_t$
    3. Calculate the pseudo-loss of $h_t$:
           $\epsilon_t = \frac{1}{2} \sum_{i \le N, j \le K}^{i,j=1} D_t(i,j) \left(1 - h_t(x_i, y_i) + h_t(x_i, y_j)\right)$
    4. Set:
           $\beta_t = \frac{\epsilon_t}{1 - \epsilon_t}$
    5. Update distribution $D_t$:
           $D_{t+1}(i,j) = D_t(i,j) \beta_t^{\frac{1}{2}(1 + h_t(x_i, y_i) - h_t(x_i, y_j))}$
    6. Normalize $D_{t+1}$
Output: final hypothesis:
           $f(x) = \arg\max \sum_t (\log \frac{1}{\beta_t}) h_t(x, y_j)$
```

- Initialize step is very clear. The algorithm gives a uniform sample weights for every sample at every class that is not the label.
- Next, for each of M estimators, we do sub step (1) (2) (3) (4) (5) (6)
- At step (1), we fit the current estimator to the training data if this is the first estimator, otherwise we resample the data based on the distribution and train.
- At step (2), we obtain a hypothesis that can output the probability of a class for a sample.

- At step (3), we compute the pseudo-loss of the hypothesis. This equation may look complicated, but it is simple. For each sample, we subtract the probability of the correct class from the probability of all the other classes and multiply this vector with the weight vector. After this, we sum up for every sample.
- At step (4), we set the model weight for the current hypothesis.
- At step (5), we update the sample weight by multiplying them with model weight to the power of the loss.
- At step (6), we normalize the weight.
- After finishing the above steps for M classifiers, the algorithm outputs the model weights and we can use them to combine predictions of all M classifiers.

5.1.2 Multiclass Adaboost algorithm implementation in Python

In this experiment, two machine learning libraries were used, namely Keras and Scikit-learn. Keras played a key role in being a framework for CNNs. Scikit-learn offered a wide range of machine learning tools and algorithms. It also provided an excellent implementation for Adaboost algorithm but unfortunately, it was neither compatible with Keras model nor implemented using the chosen algorithm. So, a fresh implementation of the algorithm was written. The program is shown in Appendix1 and may help clarifying the algorithm further.

5.1.3 Results

To see the impact of Adaboost, four experiments were conducted. In each one of the experiments, the dataset was first split into two parts. An empty model was trained on the big part in a Adaboost manner, then the pool was tested against the small part. Table 2 reports the accuracies of Adaboost algorithm:

Dataset trained on	**Original model with 1 estimator**	**Boosted model with 10 estimators**
Original (32x32)	0.82	0.84
Haar level 1 (16x16)	0.68	0.72
Haar level 2 (8x8)	0.52	0.56
Difference data (16x16)	0.64	0.67

Table 2 - Impact of 10-estimator Adaboost on different datasets

5.2 Averaging

This method is very simple. After we train multiple neural networks, we average their predictions on test set to produce the final predictions:

$$P_{final} = \frac{1}{n}\sum_{k=1}^{n} P_k \ where\ n\ is\ the\ number\ of\ models$$

5.3 Geometric mean

This method is similar to averaging, but instead taking average of predictions, we take geometric mean of them:

$$P_{final} = \sqrt[n]{\prod_{k=1}^{n} P_k} \ where\ n\ is\ the\ number\ of\ models$$

5.4 Stacking

Stacking or stacked generalization is an ensemble method that feeding probabilities from a set of estimators to another one before production final predictions which is first proposed by David H. Wolpert (Wolpert, 1992). After training four networks with each one on different data sets, they were used to generate predictions for the training set. Next, there predictions were concatenated into a matrix of 40 features. Now another model was trained on top of these predictions. The reason was that the stacking model can learn different weights for four neural networks. This contrasts with averaging which gives 0.25 to all the networks. Also, this method can reduce over-fitting if there is a model which learns too aggressively.

However, which model should be use for stacking and how one can maximize its accuracy? In this experiment, logistic regression was chosen as a stacking model because it is very fast and simple (it only has two main parameters). Also because of this simplicity, the model was tuned easily so performance was maximized. After approximately 16 minutes of thoroughly searching through possible parameters, best set of parameters were achieved: {C = 100, penalty=l2} and this stacking model scored 0.8298 on Kaggle.

5.5 Summary

Table 3 summarises the effect of different ensemble approaches:

Approach	Input	Final acc
Average	Probabilities of 4 CNNs	0.7923

Geometric mean	Probabilities of 4 CNNs	0.7921
Boosting	Probabilities of 10 CNNs along with their model weights	0.841
Stacking	Probabilities of 4 CNNs and their weights	0.8296

Table 3 - Comparison of different ensemble methods

6 EXPERIMENTS

6.1 Scope

This experiment investigates only the impact of Haar transform and model ensemble on efficiency and accuracy of deep learning models. Architecture of models, optimization algorithm or tuning parameters will be tried to be left as default or as simple as possible. This purpose is to see how big the impact of the proposed method.

6.2 Procedures

1. Download CIFAR-10 data set (Krizhevsky, 2009).
2. Reading the original images and perform pre-processing as followed:
    a. Convert colored images to grayscale images and save this data as **original_images**.
    b. Perform Haar transform level 1 and level 2 on original_images and save them as **haar_lvl1** and **haar_lvl2** respectively.
    c. Calculate the difference images of the above data sets and save it as **diff_data**.
3. For each one of these four data sets, program and train a deep learning model in Keras.
4. Apply basic tuning and validating to maximize the accuracies.
5. Report the testing errors and training time of the models.
6. Now apply model ensemble on these models and report the testing errors.
7. Answer the question whether the proposed method leads to any improvement.

6.3 Data set and tools

The image data set used in this experiment is the public CIFAR-10 collected by University of Toronto (Krizhevsky, 2009). This data set contains 60000 different coloured images of size 32x32 and they are all correctly labelled and classified into one of ten different classes.

Deep learning models and model ensemble are programmed in Python using common machine learning libraries:

- Keras for convolutional neural networks
- Scikit-learn for other Machine learning algorithms or techniques

Beside Python, MATLAB is also used since it provides excellent toolboxes for fast reading and writing images as well as performing Haar transform.

6.4 **Goals**

With this experiment, the author aims to achieve:

- Deeper understanding of Convolutional neural networks.
- Understanding of Haar image compression technique.
- Understandings of different model ensemble techniques.
- Better programming skills on implementing Machine learning models using relevant libraries and toolboxes.

6.5 **Results**

The result of the experiment is considered successful as the author meets his objective which is a broader knowledge of relevant topics. In addition, the proposed method shows promising potential because of a significant boost in accuracies of deep learning models. However, in order to be applied in practice, further consideration on matters such as training time or predicting time (especially Adaboost) must be done under case-by-case basis.

7 CONCLUSIONS

As we see, deep learning models are doing very well on classification tasks. Original dataset was still the best to train model on since it contained the most details of images. As deeper Haar transform was applied, the more details were thrown away the worse models performed. Section 3.2.2 reports the accuracies of modes when they were trained directly on these datasets. However, it has been shown that ensemble methods boost these accuracies by an amount depending on which method. There were four different methods which have been used during the experiments. Section 5.5 compares best accuracies of these methods.

Adaboost and stacking seemed to perform very well during experiments. The reasons for this was that they both try to balance the learning capability of different models. Adaboost was slow to train but showed very good promise as it often boosted the accuracies by at least 2-3 percent. Stacking also boosted but just slightly because it depended heavily on the stacking model and how we tune hyperparameters.

The two left methods are averaging and geometric mean. These methods did not boost but made it even worse if we compare to the best accuracy of individual deep learning models. It was because these two methods are two simple and they weight every models the same. Therefore, they can only score between the best and the worst.

This thesis suggests some more future works:

- Different number of estimators in an Adaboost pool can be experimented.
- Stacking depends heavily on the stacking model and only logistic regression was tried during experiments. However, there are other model can be used, for example: support vector machine or random forest. They are both robust and work well in practice.
- There is a similar ensemble method to stacking called data blending. It is said to perform slightly better than stacking since it leaks less data. But just like stacking, which model should we stack on?

REFERENCES

Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. *Proceedings of COMPSTAT'2010*, 177 - 186.

CIFAR-10 - Object Recognition in Images. (n.d.). Retrieved from kaggle: https://www.kaggle.com/c/cifar-10

CONV layer. (2016). Retrieved from datascience.stackexchange.com: https://goo.gl/ItjK9C

H. Robbins, D. Siegmund. (1971). A convergence theorem for non negative almost supermartingales and some applications. *Herbert Robbins Selected Papers*, 111 - 135.

Hinton et al. (2012). ImageNet Classification with Deep Convolutional. *Advances in Neural Information Processing Systems 25 (NIPS 2012)*.

Holger Schwenk, Yoshua Bengio. (2000). Boosting Neural Networks. *Neural Computation (Volume 12, Issue 8, August 2000)*, 1869 - 1887.

Krizhevsky, A. (2009). Learning Multiple Layers of Features from Tiny Images.

LeCun et al. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation (Volume 1, Issue 4. December 1989)*, 541 - 551.

LeCun et al. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE (Volume: 86, Issue: 11, Nov 1998)*, 2278 - 2324.

M. W. Gardner, S. R. Dorling. (1997). Artificial neural networks (the multilayer perceptron) - A review of applications in atmospheric sciences. *Atmospheric Environment (Volume 32, Issues 14-15, 1 August 1998)*, 2627 - 2636.

Rafael C. Gonzalez, Richard E. Woods. (n.d.). In *Digital image processing 3rd edition* (pp. 474-475). Person.

Samuel, A. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of research and development*.

Srivastava et al. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research 15*.

understanding-convolutional-neural-networks-for-nlp. (2015). Retrieved from www.wildml.com: https://goo.gl/J0Qdyk

Wolpert, D. H. (1992). Stacked generalization. *Neural Networks (Volume 5, Issue 2, 1992)*, 241 - 259.

Zhu et al. (2006). Multi-class adaboost. *Statistics and its Interface (Volume 2, 2009)*, 349 - 360.

APPENDIX 1 – ADABOOST IMPLEMENTATION

```
def pseudo_loss(weights, y_pred, y_true):
    """
    Calculate pseudo_loss
    """
    loss = np.transpose(y_pred) - np.sum(np.multiply(y_pred, y_true), axis=1)
    e = 0.5*np.sum(weights*np.transpose(1+loss))
    beta = e/(1-e)
    w = weights*np.transpose(beta**(0.5*(1-loss)))
    w = w / np.sum(w)   # Normalize
    return beta, w, e


def probabilities(weights):
    """
    Calculate probabilities to resample data based on weights
    """
    p = np.sum(weights, axis=1)
    return p / np.sum(p, axis=0)


def train(X, y, X_val, y_val, M):
    """
    :param X: training samples
    :param y: training labels
    :param X_val: validating samples
    :param y_val: validating labels
    :param M: number of estimators
    :return: list of trained estimators and model weights
    """
    K = 10  # number of classes
    estimators = []  # list of M estimators
    n = X.shape[0]  # number of training samples
    n2 = X_val.shape[0]  # number of validating samples
    w = (np.ones((n, 10))-y)*(1.0/(n*(K-1)))  # sample weights
    betas = []  # list of model weights

    for m in range(M):
        # Resample training data
        indices = np.random.choice(range(n), n, replace=True, p=probabilities(w))
        if m == 0:
            X_resampled = X
            y_resampled = y
        else:
            X_resampled = X[indices, :]
            y_resampled = y[indices, :]

        # Fitting
        print "\nFitting %d-th estimator" % (m+1)
        estimator = make_estimator()
        estimator.fit(X_resampled, y_resampled)

        # Validating error
        y_pred = estimator.predict(X_val)
        incorrect = y_pred != np.argmax(y_val, axis=1)
        error_rate = float(sum(incorrect)/float(n2))
        print "Accuracy is %f" % (1-error_rate)

        # Computing loss
        y_pred_proba = estimator.predict_proba(X_resampled)
        betaT, w, e = pseudo_loss(w, y_pred_proba, y_resampled)

        # Adding things
        betas.append(betaT)
        estimators.append(estimator)

    betas = np.log(1.0/np.array(betas))
    return estimators, betas
```

APPENDIX 2 – OUTPUT OF ADABOOST PROGRAM ON HAAR LEVEL 2 DATA SET

```
Fitting 1-th estimator
Accuracy is 0.520200
0.286579235402 0.401697356768

Fitting 2-th estimator
Accuracy is 0.474100
0.26847141123 0.36700057298

Fitting 3-th estimator
Accuracy is 0.480800
0.271085074961 0.371902214715

Fitting 4-th estimator
Accuracy is 0.468400
0.269984538093 0.369833999663

Fitting 5-th estimator
Accuracy is 0.475500
0.281239453672 0.391283933306

Fitting 6-th estimator
Accuracy is 0.474100
0.271369104215 0.372436999014

Fitting 7-th estimator
Accuracy is 0.463900
0.268328006913 0.366732647208

Fitting 8-th estimator
Accuracy is 0.470200
0.267180458007 0.364592430601

Fitting 9-th estimator
Accuracy is 0.483800
0.260863273245 0.352929659428

Fitting 10-th estimator
Accuracy is 0.469800
0.261700968945 0.354464733038
[ 0.91205632  1.00239187  0.98912432  0.99470102  0.93832181
0.98768739
  1.00312218  1.00897518  1.04148651  1.03714642]
Final accuracy is 0.558700

Process finished with exit code 0
```

APPENDIX 3 – OUTPUT OF ADABOOST PROGRAM ON HAAR LEVEL 1 DATA SET

```
Fitting 1-th estimator
Accuracy is 0.686100
0.0504095791351 0.0530856019896

Fitting 2-th estimator
Accuracy is 0.637000
0.0607932677578 0.0647283134489

Fitting 3-th estimator
Accuracy is 0.631800
0.0683382837181 0.0733509626121

Fitting 4-th estimator
Accuracy is 0.641100
0.0602554250285 0.0641189389471

Fitting 5-th estimator
Accuracy is 0.645100
0.0612167344739 0.0652085915056

Fitting 6-th estimator
Accuracy is 0.630000
0.0604638808931 0.0643550361327

Fitting 7-th estimator
Accuracy is 0.634900
0.0611015934134 0.0650779604957

Fitting 8-th estimator
Accuracy is 0.633400
0.0598257367351 0.0636326041594

Fitting 9-th estimator
Accuracy is 0.625700
0.0603047047659 0.0641747437406

Fitting 10-th estimator
Accuracy is 0.619700
0.0565608004455 0.0599517175799
[ 2.93584954  2.73755656  2.61249965  2.7470155   2.73016405
2.74334009
  2.73216934  2.7546293   2.74614555  2.81421575]
Final accuracy is 0.720500

Process finished with exit code 0
```

APPENDIX 4 – OUTPUT OF ADABOOST PROGRAM ON DIFFERENCE DATA SET

```
Fitting 1-th estimator
Accuracy is 0.643500
0.0649786928684 0.0694943445383
```

```
Fitting 2-th estimator
Accuracy is 0.583100
0.0689700286351 0.0740792786015

Fitting 3-th estimator
Accuracy is 0.578000
0.0740303545849 0.0799490079956

Fitting 4-th estimator
Accuracy is 0.574700
0.0739570445929 0.0798635140639

Fitting 5-th estimator
Accuracy is 0.579900
0.0689202621934 0.0740218687991

Fitting 6-th estimator
Accuracy is 0.578100
0.0688842671736 0.0739803493219

Fitting 7-th estimator
Accuracy is 0.565700
0.0660541453759 0.0707258831428

Fitting 8-th estimator
Accuracy is 0.568500
0.0707516431421 0.0761385722342

Fitting 9-th estimator
Accuracy is 0.580400
0.066734912569 0.0715069206678

Fitting 10-th estimator
Accuracy is 0.569500
0.0624873317218 0.066652253176
[ 2.6665099   2.60261943  2.52636625  2.52743618  2.60339471
2.60395577
  2.64894367  2.57520028  2.63796104  2.70826643]
Final accuracy is 0.670800

Process finished with exit code 0
```

APPENDIX 5 – OUTPUT OF ADABOOST PROGRAM ON ORIGINAL DATA SET

```
Fitting 1-th estimator
Accuracy is 0.741600
4.99829717168e-05 4.99854701392e-05

Fitting 2-th estimator
Accuracy is 0.678200
0.0499596914037 0.0525869175778

Fitting 3-th estimator
Accuracy is 0.665200
0.0498335826929 0.0524472153354
```

```
Fitting 4-th estimator
Accuracy is 0.674500
0.0503087887774 0.0529738384255

Fitting 5-th estimator
Accuracy is 0.679000
0.05042335551 0.0531008800634

Fitting 6-th estimator
Accuracy is 0.668000
0.0506258738288 0.0533255251362

Fitting 7-th estimator
Accuracy is 0.679800
0.0499913903163 0.0526220392181

Fitting 8-th estimator
Accuracy is 0.675200
0.0496624617305 0.0522577081622

Fitting 9-th estimator
Accuracy is 0.671500
0.0505178182493 0.0532056516913

Fitting 10-th estimator
Accuracy is 0.666800
0.051000476882 0.0537413092837
Final accuracy is 0.755400
Generating predictions
Writing to csv

Process finished with exit code 0
```