CASESTUDY: IMAGENET CLASSIFICATION WITH DEEP CONVOLUTIONAL NEURAL NETWORKS

Keshav Sridhar

ABSTRACT

The paper[5] utilizes a deep convolutional neural network to classify the 1.2 million high-resolution images for the ImageNet competition and also achieves state of the art test set error rates for both top-1 and top-5. The network utilizes non-saturating neurons paired with an efficient GPU implementation for the convolution operation. The paper also uses the relatively new technique of "Dropout" which strongly helped in achieving the state of the art scores for the image classification of the 1000 categories.

1. INTRODUCTION

Current state of the art Convolutional Neural Networks(CNNs) utilize efficient machine learning techniques to tackle the image classification problem. But even though they succeed in smaller object recognition tasks, they stumble when attempting to solve a larger set of classification objects within a single network because of the scarcity of larger and varied image training datasets. This problem was addressed through the release of the ImageNet[1] dataset which consists of over 15 million labeled high-resolution images in over 22000 categories. This caused a new problem due to the complexity of classifying hundreds or even thousands of classes from millions of images which called for a model that has the capacity to learn large feature sets.
Even with the efficiency of CNNs and using prior trained feature sets, their inability to scale to a problem of this size was a stumbling block. The authors of this paper had trained one of the largest ever CNNs by utilizing a highly optimized GPU implementation of the 2D convolution and all other CNN operations. They trained the network on two GTX 580 3GB GPUs, which took six days to train. The advantage of their procedure is that, by the availability of even larger datasets and faster GPUs, their results can be improved.

2. DATASET, DOWNSAMPLING, METRIC

Using Amazon's Mechanical Turk crowd-sourcing, ImageNet database was built. It has over 15 million labeled high resolution images in over 22000 categories. Every year from 2010 and onward, ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) is held with the challenge being to successfully tackle the ImageNet database with the least error rate possible. The competition utilizes a smaller subset of the original ImageNet images, and it contains 1.2 million training images, 50000 validation images and 150000 test images. They capture two main metrics, namely Top-1 and Top-5 error rates with Top-5 being the fraction of test images in which the label reported by the model is not among the five most probable images.
As ImageNet contains images of various resolutions, the model built by the authors needed a stable set resolution size. So they had to downsample to a 256x256 resolution. They trained the network on the raw RGB pixels of these images with no further preprocessing.

3. ARCHITECTURE

The architecture of the network consists of eight learned layers, out of which five are convolutional layers and three are fully connected layers as shown in Figure 2. The main features of the architecture is described in the following subsections.

3.1. Activation units

The traditional method of choice for activation functions in neural networks are either the hyperbolic tangent function($f(x) = tanh(x)$) or the logistic sigmoid function($f(x) = (1 + e^{-x})^{-1}$). But due to the nature of the problem, while backpropagating during gradient descent to learn the weights, using those functions as activation makes training more slow for these saturating non linear functions. Following Nair and Hinton[2] the authors use Rectified Linear Units (ReLUs) as a non saturating non linear function specified by,

$$f(x) = max(0, x)$$

This has an effect of accelrated learning times which in turn has a much larger influence on bigger models trained on massive datasets as is the case here(Figure 1).

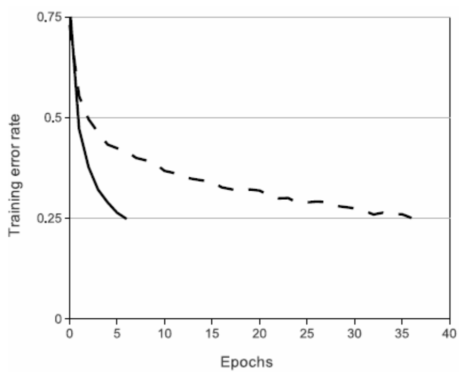

Figure 1: A four-layer convolutional neural network with ReLUs **(solid line)** reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons **(dashed line)**. The learning rates for each network were chosen independently to make training as fast as possible. No regularization of any kind was employed. The magnitude of the effect demonstrated here varies with network architecture, but networks with ReLUs consistently learn several times faster than equivalents with saturating neurons.

3.2. GPUs

The authors spread the neural network across two GTX 3GB GPUs. They do this as current GPUs integrate nicely with cross-GPU parallelization by reading and writing to one another's memory directly. Also only in some layers the GPUs need to communicate with the other. Due to this, essentially each half of the neurons are deployed on the respective GPUs. This step reduces their total error rate across both the top-5 and top-1 tasks.

3.3. Local Response Normalization

One significant advantage of using ReLUs is that they do not need their input to be normalized to prevent saturation. The authors state that even if some training examples produce some positive input then that particular neuron will learn. However, they still use local normalization to aid the model generalization. I suspect that they do this because there are chances that in some cases the ReLUs become "dead", i.e due to inactive inputs there will be no learning to back propagate and they might remain in that state constantly("Dying ReLU problem[3]"). They normalize by the following scheme,

$$b^i_{x,y} = a^i_{x,y} / (k + \alpha \sum_{j=max(0,i-n/2)}^{min(N-1,i+n/2)} (a^j_{x,y})^2)^\beta$$

where the sum runs over n adjacent kernel maps at the same spatial position, and N is the total number of kernels in the layer. The ordering of the kernel maps is of course arbitrary and determined before training begins. This sort of response normalization implements a form of lateral inhibition inspired by the type found in real neurons, creating competition for big activities amongst neuron outputs computed using different kernels. The constants $k, n, \alpha,$ and β are hyper-parameters whose values are determined using a validation set; the values after validation were: $k = 2, n = 5, \alpha = 10^{-4}$, and $\beta = 0.75$.

3.4. Pooling

Pooling is a technique that summarizes data from a group of cells in the kernel map. In traditional methods the neighborhoods summarized by different pooling units don't overlap. But for this task the authors utilized overlapping pooling, where in the different maps overlap within a certain range of cells that is specified. This was also useful in reducing their error rates. It was also observed that this techniques makes the model harder to overfit to the training data.

3.5. Overall Architecture

The first convolutional layer filters the 224x224x3 input image with 96 kernels of size 11x11x3 with a stride specified as 4 pixels. The second convolutional layer applies a filter to the first convolutional layer output with 256 kernels of size 5x5x48. The third convolutional layer has 384 kernels of size 3x3x256 connected to the (normalized, pooled) outputs of the second convolutional layer. The fourth convolutional layer has 384 kernels of size 3x3x192, and the fifth convolutional layer has 256 kernels of size 3x3x192. The fully-connected layers have 4096 neurons each. This is shown in Figure 2.

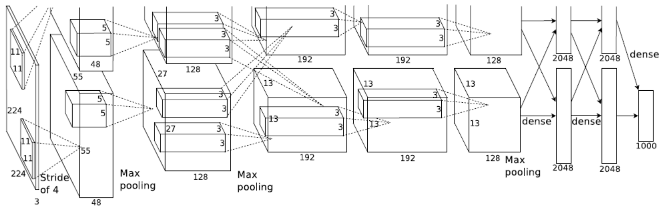

Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

4. OVERFITTING

Even though the network implemented by the authors has around 60 million parameters and the training data is well annotated with a lot of diversity, they had to consider the fact that overfitting was going to be a problem with such a large model. They combat the problem of overfitting by using two different approaches.

4.1. Augmentation

One way of reducing overfitting in a model that classifies images is to augment the data with artificial images and thus enlarge the dataset. They perform two types of transformations, namely image translations & horizontal reflections and the other way is by performing Principle Component Analysis(PCA) on the RGB pixel values to all the training data images and add them as new features. This also reduced the final error on the test set.

4.2. Dropout

In ensemble techniques, it is common to either stack models or combine multiple models in some other way to achieve better prediction results for the task at hand. But doing this in an efficient way for neural networks is a very hard problem. But a new novel technique called "Dropout" was introduced[4], in which the output of each hidden neuron is set to zero with probability 0.5. These neurons that are "dropped out" don't participate in forward and back propagation, thus in each pass the neural network learns a new architecture and is forced to learn even more robust features, with the tradeoff being that the number of iterations till convergence is doubled. Without this technique, the authors noticed that the model was being overfitted substantially.

5. LEARNING

The weights for the whole model were learnt with stochastic gradient descent with a batch size of 128 examples, momentum of 0.9 and weight decay of 0.0005(this was significant as per the authors). The update rule for the weights is given as follows,

$$v_{i+1} = 0.9v_i - 0.0005\epsilon w_i - \epsilon \frac{dL}{dw}|_{w_i,D_i}$$
$$w_{i+1} = w_i + v_{i+1}$$

where i is the iteration index, v is the momentum variable, ϵ is the learning rate, and $\frac{dL}{dw}|_{w_i,D_i}$ is the average over the ith batch D_i of the derivative of the objective with respect to w, evaluated at w_i.
The weights were initialized with a Gaussian distribution with $\mu = 0$ and $\sigma = 0.01$. The biases for the second, fourth, fifth convolutional layers and the fully connected hidden layers were initialized with the constant 1 to accelerate the early stage of learning by providing the ReLUs with positive inputs. In remaining layers the bias was set to 0. ϵ was initialized with 0.01. The authors followed a heuristic for the learaning rate ϵ, which was to divide the learning rate by 10 whenever the validation error rate stopped improving. They had trained the network for 90 cycles through the whole training set of 1.2 million images.

6. CONCLUSION

This paper proved that using a deep convolutional neural network, state of the art results can be achieved even for challenging and huge datasets using only supervised learning. This network was finely tuned and configured to tackle this specific task and thus the authors note that the network configuration needs to be as exactly as specified in the paper, else the performance rapidly degrades even if a single convolutional layer is removed. The authors goal in the future is to use very large and deep neural networks on video sequences, where the temporal element could offer useful information that is not readily apparent in still images.

7. REFERENCES

[1] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In CVPR09, 2009.
[2] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proc. 27th International Conference on Machine Learning*, 2010.
[3] https://en.wikipedia.org/wiki/Rectifier_(neural_networks)
[4] G.E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R.R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.
[5] A. Krizhevsky, I. Sutskever, G.E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems 25 (NIPS 2012)*

CASESTUDY: SUPPORT VECTOR MACHINES FOR HANDWRITTEN NUMERICAL STRING RECOGNITION

Keshav Sridhar

ABSTRACT

Handwritten numerical string detection pose a different challenge compared to isolated digit recognition. Neural networks have been widely used in both string as well as digit recognition. The paper[4] utilizes support vector machines to tackle this problem. Due to combined digits, over segmentation and under segmentation can affect recognition rates. By experimental results SVMs are shown to be better at handling these issues compared to state of the art multi-layer perceptron neural networks.

1. INTRODUCTION

Support Vector Machines (SVMs) have recently gained a lot of traction in the machine learning community because of its ability to be applied in a variety of subject areas successfully. It is able to generalize even in high dimensional spaces and is shown to be able to beat neural networks employing empirical risk minimization principle. But their drawback has been the training and running time compared to the neural networks, is slower. But more research on this has helped push SVMs even further with the advent of the kernel trick and other improvements for scale and speed. With this in hand, the paper addresses the more complex issue of handwritten numerical string recognition.

2. PROBLEMS IN HANDWRITTEN DIGIT STRING RECOGNITION

As mentioned earlier, previous work had utilized SVMs only on isolated digit recognition and in the case of digit string recognition, the state of the art technique uses a segmentation based approach with Multilayer Perceptrons as both classifiers and verifiers. This approach combines outputs from different levels such as segmentation, recognition and post processing via a probabilistic model which integrates nicely. Figure 1 explains the general architecture of the system.

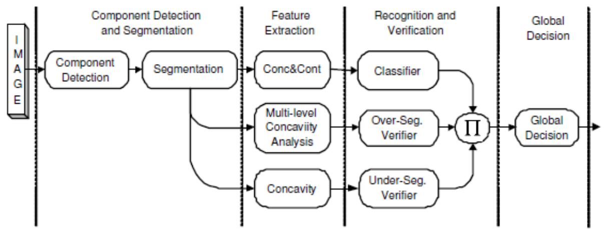

Figure 1. Block diagram of the digit string recognition system.

This system provides good results but has to deal with outliers, which in this case are over- segmentation and under-segmentation. MLPs have difficulty when it comes to dealing with this kind of outlier which is common in a string detection system. To ease this, the system uses verifiers. Without these verifiers to detect outliers, the system performs poorly, because the segmented parts could be identified as separate digits as shown in Figure 2.

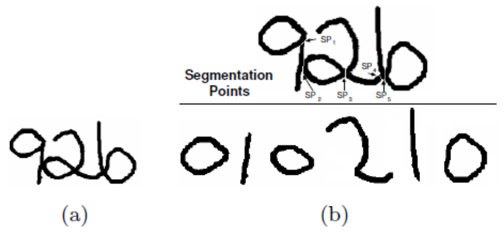

Figure 2. Example of over-segmentation: (a) Original string and (b) over-segmented pieces.

3. SVM

SVMs aim is to find an optimal hyperplane without depending on some kind of probability estimation. This hyperplane is a linear decision boundary that separates the two classes with the largest margin possible. This method was introduced by Vapnik[1]. This method uses a fraction of data points called support vectors. Thus the probability of making an error also depends on these vectors. It depends on the assumption that

the classes are linearly separable.
An extension can be made to fit non-linear decision boundaries using a method called as a Kernel. In this method the feature space is of a higher dimensionality than the input space and it is in this space that the optimal hyperplane will be fitted. We can formulate the decision function as a Kernel function($K(x, x_i)$) of the input that will be used to classify the training example as follows:

$$f(x) = \sum_i \alpha_i y_i K(x, x_i) + b \ldots (1)$$

where, the parameters α_i and b are found by quadratic programming(subject to constraints) and y_i is the label of the input sample x_i.
While finding these parameters, we should also consider the tradeoff parameter C, which penalizes the errors, i.e a soft-margin. This parameter can have a huge impact on the performance of the classifier. If the errors are punished by a lot, then the SVM can overfit the training data.
Also traditionally, SVM is a binary classifier. Thus we need to be able to extend it to be able to deal with multi class classification($q-$class problem). The two main approaches are one-against-others and the pairwise method. In the pairwise method, binary SVM classifiers are placed in a tree structure and as we go up the tree, each classifier beats the other and we will have one left at the root. In this way, we need to train $\frac{q(q-1)}{2}$ classifiers for q classes(Figure 3).

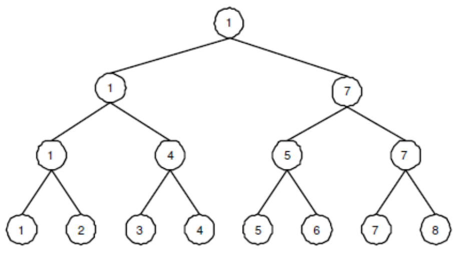

Figure 3. Example of pairwise SVM. The numbers 1-8 encode the classes.

For the one-against-others strategy, an SVM classifier is built for each class and an expert F is used to arbitrate between the outputs of each SVM class. But this introduces a new problem that it assumes all the classes are on the same scale, which need not necessarily be true. So they will have to be normalized. Argmax is a popular expert function given as follows,

$$F = argmax(s(h))$$

where $s(h)$ is the normalized output of an individual SVM classifier and h represents the set of hypotheses of each SVM classifier given as $h = (h1, \ldots, h_Q)^T$.

4. PROBABILITY ESTIMATION WITH SVM

SVMs don't produce values that are a probaility. But for the task at hand, it would be useful to have a classifier that produces a posterior probability as the baseline system illustrated in Figure 1 was built on a probabilistic framework. A Bayesian technique was proposed by Sollich[2] to estimate probabilities by SVM and also to tune hyper-parameters. This method converts SVMs as a maximum a posteriori solution to inference problems with Gaussian process priors.
Platt[3] suggested a modified logistic function to address the same given as follows,

$$P(y = 1|f(x)) = \frac{1}{1+\exp(Af(x)+B)} \ldots (2)$$

The parameters A and B is equation (2) are found by minimizing the negative log likelihood of the training data, by using a cross-entropy error function.

5. EXPERIMENTS

The paper utilizes the same architecture presented in Figure (1) where the system is composed of three main modules namely, the classifier, over-segmentation verifier and the under-segmentation verifier. The classifier classifies the digits(0-9) and the verifiers are present to detect the outliers. These are combined in a probabilistic framework. The technique demonstrated in the paper used MLPs for the verifiers but changed the classifier to an SVM with a Gaussian kernel. So 10 SVMs(one-against-rest technique) were trained on the data(NIST SD19 dataset). Using cross-validation, the parameters finalized were $\sigma = 1.15$ and $C = 1000$.

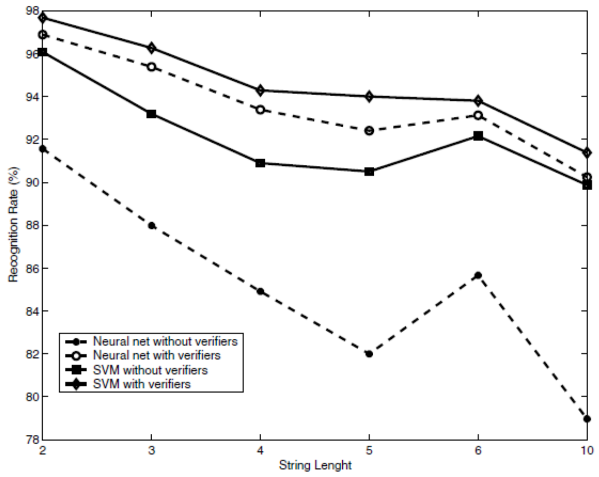

Figure 4. Comparison between the SVM- and neural-net-based systems.

As shown in Figure (4), the SVM based system both with and without verifiers outperforms the state of the art MLP based system. Also intriguing is that the SVM based systems are pretty close where as the neural net system without verifiers performs really poorly compared to the rest. The neural-network based system outperforms the SVM based system in the speed department during test phase. Speed of training for large datasets is still an ongoing issue for SVMs.

6. CONCLUSION

Much of the current literature effort was put into digit recognition and not on digit strings. This paper demonstrated via experiments that the strategy of using a one-versus-rest SVM with Gaussian kernel following Platt's methods can surpass the baseline currently held by MLP based system. The other point illustrated is the ability of SVMs to better handle outliers(segmentation issues) compared to neural nets.

7. REFERENCES

[1] V. Vapnik. *The nature of statistical learning theory*. Springer Verlag, 1995.
[2] P. Sollich. Bayesian methods for support vector machines: Evidence and predictive class probabilities. *Machine Learning*, 46(1-3):2152, 2002.
[3] J. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In A. S. et al, editor, *Advances in Large Margin Classifiers*, pages 6174. MIT Press, 1999.
[4] Luiz S. Oliveira and Robert Sabourin. Support Vector Machines for Handwritten Numerical String Recognition.*Proceedings of the 9th Intl Workshop on Frontiers in Handwriting Recognition (IWFHR-9 2004) 0-7695-2187-8/04 $20.00 2004 IEEE*

CASESTUDY: A HIDDEN MARKOV MODEL-BASED ACOUSTIC CICADA DETECTOR FOR CROWDSOURCED SMARTPHONE BIODIVERSITY MONITORING

Keshav Sridhar

ABSTRACT

Biodiversity monitoring is an intriguing problem in the modern age. Using crowdsourcing technique the authors have built a smartphone application that can detect an endangered species mating call(in this case the "New Forest cicada")[1]. Current systems in the acoustic domain utilize some form of batch signal processing for insect classification. But for real time analysis, this is not computationally feasible. So, the authors have built a model based on a hidden markov model to which they feed two frequency feature vectors which they extract with the help of Goertzel's algorithm[2]. They show that this is much more resitant to noise as well as being computationally efficient compared to the batch insect classification.

1. INTRODUCTION

Many countries are seeing the benefits of monitoring biodiversity as it is a key measure of the health of an ecosystem, and for land-use and climate change impact on the natural environment. The authors are interested in a form of cicada that is native only to the UK, namely the cicadas from the New Forest national park on the south coast of England. Traditional approaches like hiring experts to manually survey and monitor endangered species are being curtailed due to high costs and scarcity of trained experts and thus automated means are being explored.

Fixed sensors with sensitive microphones are deployed into the environment. These sensors record the sounds that the insects emit and these recordings are then analysed later to automatically identify the insects whose calls were heard. These signals are modeled as a time signal coding problem and they are handled by Hidden markov models. In 2012, Chaves et.al[3] presented a state-of-the-art approach that preprocesses the recorded sound to remove unsounded periods where no insect call is detected, that maps the raw frequencies to the mel scale, which better represents human hearing; then it converts it back into a pseudo-time domain, called the cepstrum, by calculating a number of mel frequency cepstral coefficients (MFCC), that are used as features for the HMM classification with just one state per species.

It helps that the cicadas being monitored have a particularly high-pitched mating song which can be easily detected by microphones. As the forest being monitored sprawls a huge area(600 km^2), it does not make sense to spread sensors all around the forest. Thus the authors pursue a crowdsourced technique, where in visitors to the national park can install an app that can classify cicadas and record further information. To do this they need the algorithm to run in real-time on a smartphone on limited computational resources. It cannot be a batch algorithm as the user's need to get immediate feedback so that they know as to when not to upload recordings and when to proceed to get the best quality recordings of the new forest cicadas.

The authors use the Goertzel algorithm, which is an efficient method for approximating individual terms of a discrete Fourier transform (DFT) [Goertzel, 1958] to calculate the magnitude of two specific frequency bands; one centred at 14 kHz, corresponding the central frequency of both the insects calls, and one centred at 8 kHz, which is far from both general background noise and the insects call. We use the ratio of these magnitudes as a single feature, which identifies the song of either the bush cricket or the New Forest cicada. Then, they use a four-state hidden Markov model that explicitly represents both the idle, un-sounded period between insect calls, and also the short pauses between the chirps of the bush crickets song. Hence, rather than attempting to recover the time domain information lost while removing unsounded periods through heuristic methods, they explicitly capture this in the HMM, as this approach can be readily extended to cover insect calls of more complexity, all within the same principled framework. They finally use the Viterbi algorithm to identify the most likely sequence of insect calls at any point in time.

2. PREPROCESSING

In the preprocessing stage, the authors efficiently extract individual terms of a discrete Fourier transform from the raw audio recordings using the Goertzel algorithm. Out of these individual terms, the authors have found a combination of two of them to be robust against noise. They use this combination as a new feature input to a hidden markov model with four states which corresponds to different periods of a recording. Using this, they postulate that this approach can also be used for many different insects, albeit with a few modifications.

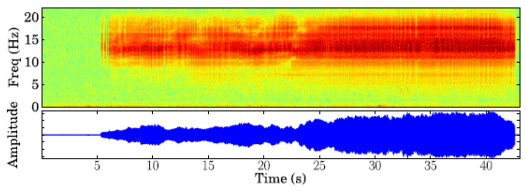

Figure 2: Spectrogram and waveform of a New Forest cicada call (recording by Jim Grant, 1971 and courtesy of the British Library, wildlife sounds collection).

Goertzel's algorithm is explained as follows,

An efficient implementation of the Goertzel algorithm requires two steps. The first step produces a coefficient that can be pre-computed and cached to reduce CPU cycles:

$$c = 2\cos\left(\frac{2\pi f}{f_s}\right) \quad (1)$$

where f is the central frequency in question and f_s the sampling rate of the recording.

The second step consists of iteratively updating the values of a temporary sequence y with any incoming sample s such that:

$$y_n = hamming(s) + (c \cdot y_{n-1}) - y_{n-2} \quad (2)$$

where the samples are passed through a *Hamming* filter, given by:

$$hamming(s) = 0.54 - 0.46\cos\left(\frac{2\pi s}{N-1}\right) \quad (3)$$

and the length of the sequence of samples N determines the bandwidth B of the Goertzel filter, such that:

$$B = 4\frac{f_s}{N} \quad (4)$$

A sequence length N yields larger bandwidth, at the cost of a noisier output. In practice, we use multiples of 64 samples to match a typical smartphone's buffer size. For example, a block size $N = 128$ samples gives a bandwidth of just under 1.4 kHz. The magnitude m of the frequency band centred at f and with bandwidth B is then given by:

$$m_f = \sqrt{y_N^2 + y_{N-1}^2 - c \cdot y_N \cdot y_{N-1}} \quad (5)$$

Compared to the fast Fourier transform (FFT), which has a complexity of O(NlogN), the Goertzel algorithm is only computed in order O(N), where N is the number of samples per window. Also, the sample update described in equation (5) can be processed in realtime, so it doesn't require a separate background thread on the smartphone app, thus saving computation power.

3. HIDDEN MARKOV MODEL

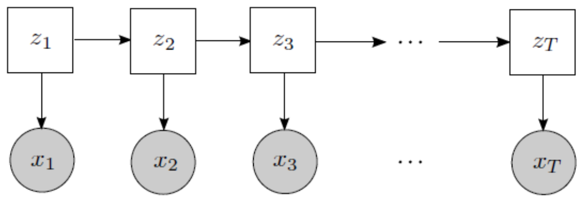

Figure 3: A hidden Markov model. Unshaded square nodes represent observed discrete variables, while shaded circular nodes represent hidden continuous variables.

In a hidden markov model each node represents a hidden state which emits some output. In sich a model, there are sets of probabilities for each component od the model. There are initial probabilities which define the probability of being in one of the four hidden states at the start time of the recording, then there are transition probailities which give information about state to state transfer and finally emission probailities which give the probabilities of a particular hidden state emitting a particular signal(in this case bush crickets and the ciacdas song). Figure 3 shows the general architecture of a hidden markov model.

The joint likelihood of a hidden markov model is given as,

$$p(\mathbf{x},\mathbf{z}|\boldsymbol{\theta}) = p(z_1|\boldsymbol{\pi})\prod_{t=2}^{T} p(z_t|z_{t-1},\mathbf{A})\prod_{t=1}^{T} p(x_t|z_t,\phi)$$

where π is the initial state probabilities, $\mathbf{A}$ represents the transition state probabilities and $\phi = (\mu, \sigma^2)$ represents the emission probabilities of observed feature $\mathbf{x}$. z_t represents a hidden state.

The authors HMM has four states in which one state is the idle state(I) where there is no insect singing, a cicada singing state (C), a state where the dark bush cricket is chirping (BC) and a short pause in between the dark bush crickets chirps (BSP).

The model parameters μ and σ^2 are learned empirically for each state k using:

$$\mu_k = \frac{\sum_{t=1}^{T}\ln(x_t)}{T}, \sigma_k^2 = \frac{\sum_{t=1}^{T}(\ln(x_t) - \mu)^2}{T}$$

The model is shown in figure 5:

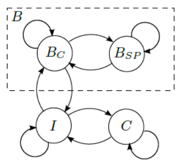

Figure 5: Four-state finite state machine. B_C represents the bush cricket's chirp, B_{SP} represents the short pause between chirps, I represents the idle state and C represents the cicada's song.

4. RESULTS

The authors use precision, recall and the combined harmonic mean, i.e the F-1 score to assess their technique versus the one proposed by Chavez et. all. The results are shown in table 1 as follows,

Approach	**Precision**	**Recall**	F_1**-score**
Our approach	1.000	0.914	0.955
Chaves *et al.* [2012]	0.563	0.071	0.126

Table 1: Accuracy metrics of cicada detection

5. CONCLUSION

The authors have presented a novel approach that is a real game changer in insect classification as previous appraches were neither very accurate nor computationally fast due to their complexity. They have beaten the state of the art results by a large margin. The main advantage of their approach is that almost all other techniques do a batch process of a variety of insects and tries to classify a lot of them where as the approach presented by the authors narrows the insects down to two species and in turn it becomes a dual class problem.

6. REFERENCES

[1] Davide Zilli, Oliver Parson, Geoff V Merrett, Alex Rogers, A Hidden Markov Model-Based Acoustic Cicada Detector for Crowdsourced Smartphone Biodiversity Monitoring, *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*
[2][Goertzel, 1958] G. Goertzel. An algorithm for the evaluation of finite trigonometric series. *The American Mathematical Monthly*, 65(1):3435, 1958.
[3] [Chaves et al., 2012] V. A. E. Chaves, C. M. Travieso, A. Camacho, and J. B. Alonso. Katydids acoustic classification on verification approach based on MFCC and HMM. *Proceedings of the 16th IEEE International Conference on Intelligent Engineering Systems (INES), pages 561566, 2012*.

CASESTUDY:
NOT SO NAIVE ONLINE BAYESIAN SPAM FILTER

Keshav Sridhar

ABSTRACT

In the age of the internet, spam filtering is a key topic being addressed regularly and improved upon through machine learning. Content-based filters are one of the most reliable ways to combat spam. The authors of the paper improve upon the traditional online naive bayes classifier with additional enhancements. They propose two enhancements, one from a theoretical point of view and one from an engineering point of view.

1. INTRODUCTION

The amount of spam over email nowadays is ever increasing and reports suggested by the authors reveal that over 90% of email traffic coming out of North America, Europe and Australasia is spam. As spam filters were being rolled out, the spammers have themselves gotten smarter in beating these filters in different ways and they are able to beat the conventional “static” methods of spam filtering easily.
Content-based analysis is shown to have good accuracy in dealing with these spammers. In content based analysis the problem of spam detection is converted as a special binary text classification problem. This is applied in many real time spam filtering systems.

2. PROBLEMS IN OTHER LEARNING METHODS

Using content analysis, spam filters have improved upon vastly in the past few years. The choice of the learning algorithm however is up for debate. In theoretical settings, many machine learning algorithms such as, Support Vector Machines, Logistic Regression, and Dynamic Markov Compression claim to get more accuracy on spam filters than the traditional Naive Bayes(NB). But the reason NB is so successful is because it is easy to code, it is faster to train and can be very adaptive to any organization’s specific needs, where as the other machine learning techniques take time to train and have some kind of memory or other constraints.

3. TRADITIONAL NAIVE BAYES

A Naive Bayes classifier follows on from the assumption that given a set of features $\chi = (X_1, X_2, ..., X_n)$ and a class C, each feature is conditionally independent,

$$P(\chi, C) = P(C) \prod_{i=1}^{n} P(X_i|C) \quad (1)$$

Applying the Bayes theorem to the spam filter domain, we get the following,

$$P(Spam|\chi) = P(Spam) \prod_{i=1}^{n} P(X_i|Spam)/P(\chi)$$
$$P(Ham|\chi) = P(Ham) \prod_{i=1}^{n} P(X_i|Ham)/P(\chi)$$

As we can just calculate the ratio of the two probailities, we need not calculate the $P(\chi)$.

$$\frac{P(Spam|\chi)}{1-P(Spam|\chi)} = \frac{P(Spam)\prod_{i=1}^{n} P(X_i|Spam)}{P(Ham)\prod_{i=1}^{n} P(X_i|Ham)} \quad (2)$$

If the odds represented in equation (2) exceeds some ratio, then we can classify the email as spam. The prior probabilities are estimated from the data and likewise the conditional probabilities as well. In case of zeros, a smoothing parameter is introduced to avoid division by zero.

In a real word setting, to make spam filter more practical, the filter must have the ability to adapt to spammers changing tactics. So spam filtering is typically deployed and tested in an online setting, which proceeds incrementally. The filter classifies a new example, is told if the prediction is correct, updates the model accordingly, and then awaits a new example.

4. NOT SO NAIVE BAYES(NSNB)

Traditional naive bayes hypothesizes that all features are conditionally independent. More advancements have been made where in instead of the original independence, a bayesian network is built. But even this, in large scale when there are usually around 10^{-4} to 10^{-6} features in spam filter domain, it makes the graph extremely huge and makes the problem intractable.
The authors make a modification of the chain rule as follows,
$P(X|C) = P(X_1, X_2, ..., X_n|C)$
$= P(X_1|C)P(X_2, X_3, ..., X_n|X_1, C)$
$= P(X_1|C)\theta(X, X_1, C)P(X_2, X_3, ..., Xn_|C)$
where θ is drawn from an unknown distribution. β_i is introduced to represent the set of features that is to be expanded before the feature X_i.
$P(\chi|C) = \prod_{i=1}^{n} P(X_i|C)\theta(\chi, C, X_i, \beta_i)$
Here it is literally hard to calculate θ directly, so the authors had made another approximation as follows,
$P(\chi|C) = \prod_{i=1}^{n} P(X_i|C)\theta(C, X_i) \quad (3)$

To add to traditional naive bayes, The authors assign two parameters to each feature namely, ham confidence, and spam confidence, and for a given feature f, the ham confidence approximates $\theta(ham, f)$, while the spam confidence approximates $\theta(spam, f)$. Before online learning, all the confidence parameters are set to 1, if mistake occurs, the system deduces the corresponding confidence parameter. In a single classification process, the authors use equation (3) instead of equation (1). To control changes of the confidence parameters, the authors add further two supportive parameters α and β, which are learning rates. If a ham mail is misclassified, then,
$ham\_confidence = ham\_confidence * \beta$,
while if a spam mail is misclassified, then
$spam\_confidence = spam\_confidence * \alpha$
They combine these two into one single factor variable called as cf, given as,
$cf = \frac{spam\_confidence}{ham\_confidence}$
So in the online process, if ham mail is misclassified then,
$cf = cf/\beta$
and if a spam mail is misclassified then,
$cf = cf * \alpha$
Using this the authors do obtain more stronger results on large benchmark corpus as depicted in Table 1 as follows,

	Spamassassin	CRM114	Bogofilter	NSNB
score	0.0590	0.0420	0.0480	0.0079

Table 1: Results on TREC 2005 Spam Track corpus, while the first three filters are traditional Naïve Bayes filters. Score reported is (1-ROCA)%, where 0 is optimal.

Besides the introduction of confidence factor and learning rate which boosts the traditional NB from theorys point of view, the authors also propose three other engineering ways to build a more robust and accuracy classifier:
• Unlike traditional smoothing technique, they choose a smooth parameter small enough.
• They introduce the logistic function and a scale parameter to scale the results.
• Use a thick threshold to adjust learning model.

5. EXPERIMENTS

In order to avoid underflow due to too many multiplications, the authors modify the ratio and make it into a log of odds ratio,
$$log \frac{P(spam)}{P(ham)} = \sum_{i=1}^{n} log \frac{P(X_i|spam)\theta P(spam, X_i)}{P(X_i|ham)\theta P(ham, X_i)} \quad (4)$$

In fact some features only appear in spam email, so the above equation is not always calculable. By applying the lidstones law, they introduced a smooth parameter ϵ, and instead of calculating the above equation, they calculate
$\frac{N(spamemailincludesX_i)+\epsilon}{N(hamemailincludesX_i)+\epsilon} * \frac{N(ham)+2\epsilon}{N(spam)+2\epsilon}$
The smoothing parameter is usually set to 1 or 0.1, but the authors chose it much smaller and found a good fit as depicted in Figure 1.

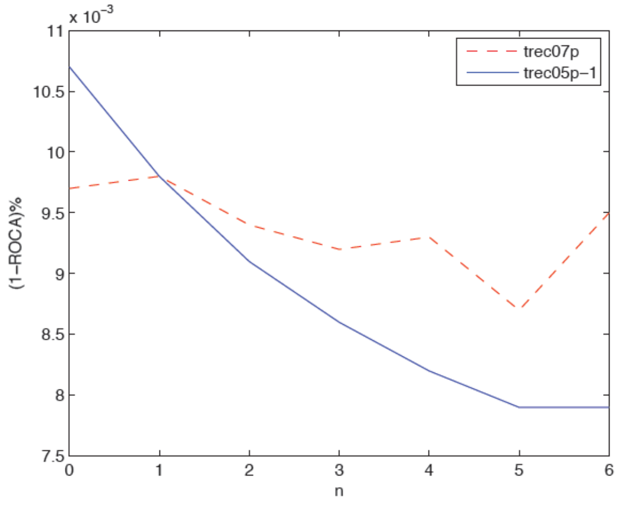

Figure 1: The smooth parameter $\varepsilon = 10^{-n}$. We can see $n = 5$, i.e. $\varepsilon = 10^{-5}$ is optimal.

6. ALGORITHM

```
Input: Dataset $U = (\mathcal{X}_1, y_1), \ldots, (\mathcal{X}_n, y_n)$, learning
       rate $\alpha$, $\beta$, thick threshold $\epsilon$
Set all features' cf to 1;
for each $\mathcal{X}_i \in U$ do
    calculate the score $p_i$;
    if $p_i > 0.5$ then
        report spam;
    end
    else
        report ham;
    end
    while $y_i$ is ham and $p_i > (0.5 - \epsilon^-)$ do
        train $\mathcal{X}_i$;
        for all features in $\mathcal{X}_i$ do
            feature's cf $\times =$ $\alpha$;
        end
    end
    while $y_i$ is spam and $p_i < (0.5 + \epsilon^+)$ do
        train $\mathcal{X}_i$;
        for all features in $\mathcal{X}_i$ do
            feature's cf $/ =$ $\beta$;
        end
    end
end
```
Algorithm 1: Pseudo-code for online learning

7. CONCLUSION

The authors improved upon the traditional naive bayes by introducing a confidence factor to each feature, and in the online process, the confidence factor is changed when an email is not well classified. Throughout this process, the Naive Bayes model gradually gets intelligent, i.e. it no longer needs the conditional independent hypothesis after sufficient iterations. The introduction of the confidence factor does bring about better results, faster training time and lesser memory consumption. This technique can also be used in other situations where Bayes theorem holds but the result is hard to calculate as the relation of variables is too complex. During the online process, we also developed other techniques such as smoothing, result scale, and thick threshold to enhance the final results.

8. REFERENCES

[1] Baojun Su, Congfu Xu. Not so Naive Online Bayesian Spam Filter,*Proceedings of the Twenty-First Innovative Applications of Artificial Intelligence Conference (2009)*