Technische Universität Berlin

Deep Neural Networks for Sound Type Classification

Author:
Xiaowei Jiang

Supervisor:
Prof. Dr. Klaus Obermayer
Dr. Johannes Mohr

28. September 2016

Versicherung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eingenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Symbolverzeichnis

AMS Amplitude Modulation Spectrogram

CNN Convolutional Neural Network

DNN Deep Neural Network

IBM Ideal binary mask

$LASSO$ least absolute shrinkage and selection operator

$NIGENS$ NI General Sound

ROC Receiver Operating Characteristic

Zusammenfassung

Soundtyp Klassification ist ein Modul vom Two!Ears system. Es gilt als ein multi-label Klassifikationsproblem. Deep neural network(DNN) ist eine Lösung für dieses Klassifikationsproblem. In dieser Bacheloraribeit wird Soundtyp Klassificationsmodels in DNNs aufgebauen. Einige Methode werden angewandt, um das Overfittingsproblem zu kontrollieren.

Abstract

Sound type classification is a module of the Two!Ears system. It can be seen as a multi-label classification problem. Deep neural networks(DNNs) provide a solution for this classification problem. In this thesis, sound type classification models of the Two!Ears system will be constructed in DNNs. Also some measures were carried out to conquer the overfitting problem.

Acknowledgements

I want to thank Dr.Johannes Mohr and Youssef Kashef. They helped me a lot during the research period and the writing of this thesis. I want to thank Prof. Klaus Obermayer. I took Machine Intelligent I and II by him. These two courses build the knowledge base of machine learning for me.

Inhaltsverzeichnis

Kapitel 1

Background and Motivation

Nowadays neural networks and deep learning has become a hot topic in pattern recognition, speaker identification, image processing and many other fields. In this thesis, deep neural networks are used as a solution for sound type classification problem. In Chapter 1 some basic concepts related to sound type classification and neural networks are introduced. This thesis is based on part of works of the Two!Ears system. Sound type classification is a module of the Two!Ears system. These basic concepts are introduced in section 1.1. Section 1.2 explains what DNNs are and introduces the Caffe framework, which is used for producing the experiment results in this thesis. After that, the main part of the thesis starts by exploring the architecture of neural networks applied to sound type classification. Three types of architectures are discussed in Chapter 2. A good architecture doesn't solve the problem completely, overfitting is always a problem one should consider in machine learning problems. In Chapter 3, data augmentation is introduced to deal with the overfitting problem. Tuning parameters of deep neural networks requires a broad and lengthy exploration of the parameter space. At last, a conclusion is drawn in Chapter 4. I will also talk about future works related to this topic.

1.1 Sound Type Classification

In this thesis, sound type classification is a supervised machine learning problem, which serves as a module of the Two!Ears System. The training data set is the NIGENS(NI General Sound) Database. In this section I will explain the sound type classification based on the six steps of supervised learning problem[1]

- Determine the type of training examples.

- Gather a training set.
- Determine the input feature representation of the learned function.
- Determine the structure of the learned function and corresponding learning algorithm.
- Run the learning algorithm on the gathered training set.
- Evaluate the accuracy of the learned function on the test set.

1.1.1 NIGENS Database

NIGENS is short for 'NI General Sound'.[2] NIGENS was compiled from the Stockmusic[1] sounds archive. The database consists of 11 classes of everyday sounds(alarm, crying baby, crash, barking dog, running engine, female speech, burning fire, footsteps, knocking on door, ringing phone, piano) and a class of general 'anything else'. For each of these 11 classes of everyday sounds, 50 wav files were selected. For the general sound class, 237 wav files were selected. Briefly, the NIGENS database is constructed from 787 high quality sound files.

In this thesis, the NIGENS Database is used as the training dataset for sound type classification.

1.1.2 The Two!Ears System

The Two!Ears system works on auditory modeling by a systemic approach.[2] The goal of this project is to develop an intelligent, active computational model of auditory perception and experience in a multi-modal context. In order to construct such a system for dynamic interaction application, a dynamic auditory scene analysis model is needed. In this model, sound type classification extracts one of the important attributes, the sound type, of corresponding aural objects.

In speak of determining the input feature representation for the sound type classification module of the Two!Ears System, rate maps, spectral features, amplitude modulation spectrogram features and onset strengths have all been used as input feature representation in previous work. These features are extracted using the auditory front-end in the Two!Ears system. However

[1]Stockmusic is the online redistributor of the famous Sound Ideas sounds library, offering individual files rather than precompiled collections. See http://www.stockmusic.com/sound_effects

[2]The official website of the Two!Ears System: http://twoears.eu/

the full combination of all these features may contain redundant information and it requires a long training duration. In this thesis, the combination of rate maps and AMS(Amplitude Modulation Spectrogram) features are determined as the input feature representation to shorten the training time.

Rate maps

Rate maps are biologically inspired spectrogram-like maps over time and frequency domain, which are supposed to represent auditory nerve firing rate. Rate maps are computed by smoothing the corresponding inner hair cell signal representation with a leaky integrator. The rate maps features used in this thesis are extracted from 16 frequency channels for each of the 63 time frames.

The rate maps extracted from NIGENS together with the target labels are stored in HDF5 files. The rate maps for all the training segments are stored as a multidimensional array with size $N \times 1 \times 63 \times 16$. N is the number of training sound sources. 63 indicates the length of sound sources in time frames and 63 time frames is a one second sound block. 16 is the number of frequency channels.

AMS features

AMS is the abbreviation for 'Amplitude Modulation Spectrogram'. The human auditory system is able to focus on a desired target source and to segregate it from interfering background noise. Currently the ideal segregation of noisy speech, as represented by the IBM(Ideal binary mask), was estimated by only exploiting AMS features[3]. Instead of linearly-scaled modulation filters, an auditory-inspired modulation filterbank with logarithmically-scaled modulation filters are used here. Each frequency channel of the auditory spectrogram was processed by a second-order low-pass filter with a cutoff frequency of 4 Hz and 8 second-order band-pass filters with center frequencies spaced logarithmically between 8 and 1024 Hz, altogether representing a modulation filterbank, which produces the final set of 9 logarithmically-scaled AMS features for each frequency channel[2].

The AMS features extracted from NIGENS together with the target labels are stored in HDF5 files. The AMS features are stored as a multidimensional array with size $N \times 9 \times 63 \times 16$. Similar to the rate maps array, N is the number of training sound sources. 9 is the number of modulation filters. 63 represents the length of sound sources in time frames. 16 indicates the number of frequency channels.

LASSO Binary Classifier

With previously mentioned features, it is quite easy to construct a binary classifier. In previous stage of the project, some approaches are developed to achieve this sound type classification task. One of the identification pipeline looks like this:

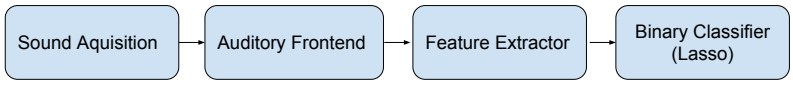


Abbildung 1.1: Identification Pipeline

In this pipeline, 11 binary one-against-all classifiers using LASSO regression method are trained to accomplish this task. For each of these 11 classes of sound types, a corresponding classifier takes the previously mentioned features $\underline{\mathbf{x}}$ as input and outputs a binary value $y \in \{0, 1\}$, where 1 represents target label. With all of the 11 binary outputs, we can construct a binary vector. An all-zero vector means the sound source is classified as 'anything else' class. This method enables multi-label classification, which means a sound source may be classified as a mix of several classes.

The accuracy performance of LASSO binary classifier is displayed in Tab.1.2.[3]Here we use sensitivity, specificity and balanced accuracy to evaluate the classification performance. These three performance metrics are calculated according to the formular below:

$$sensitivity = \frac{\#TruePositive}{\#Positive}$$

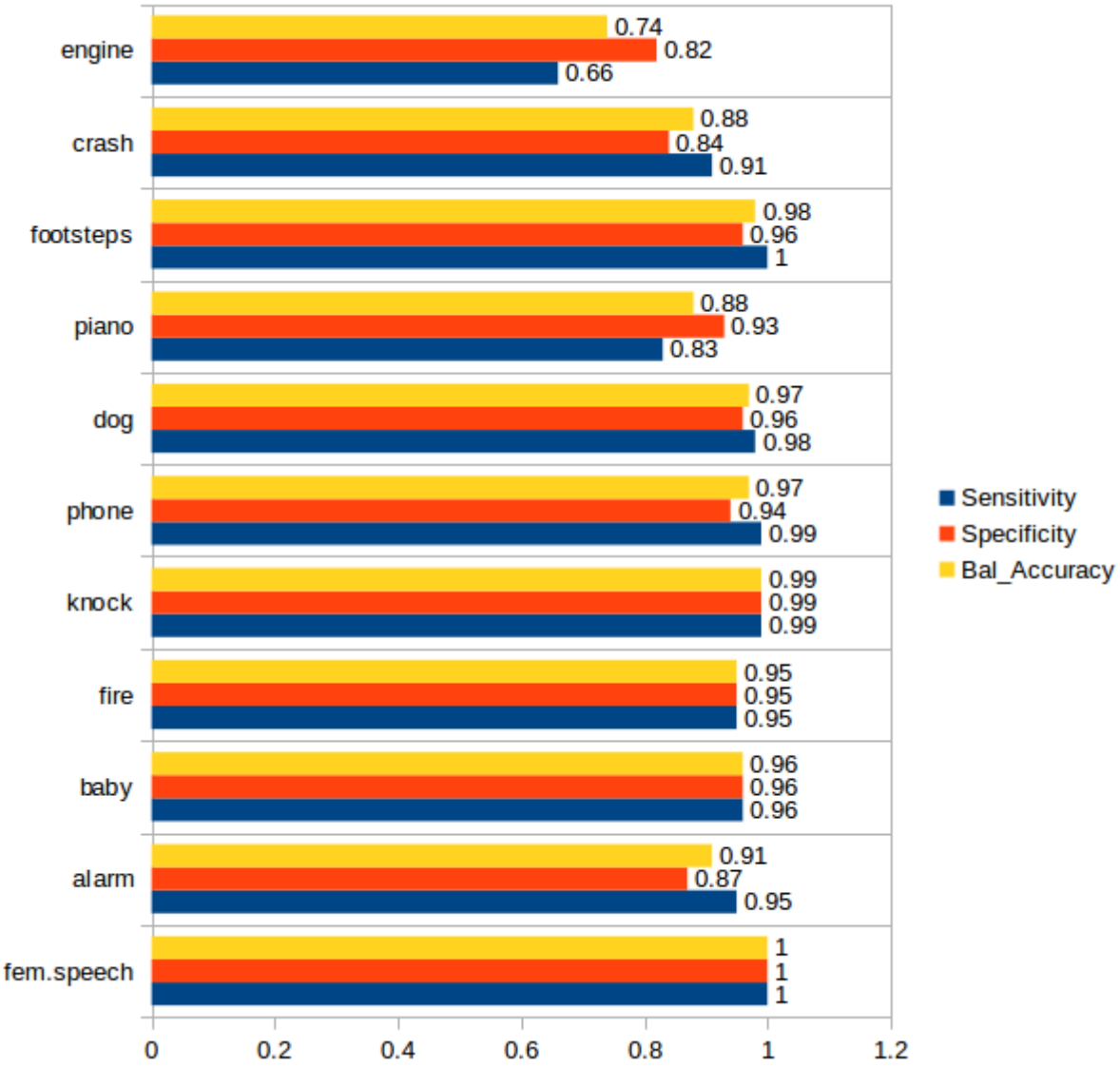
$$specificity = \frac{\#TrueNegative}{\#Negative}$$

$$balanced \quad accuracy = \frac{sensitivity + specificity}{2}$$

As we can see from the bar chart, Fig.1.2, LASSO classifier performs quite well for class 'female speech', where it achieves 100% sensitivity and 100% specificity. However, the performance of this binary classifier is not so satisfying at class 'engine' with 66% sensitivity and 82% specificity.

[3]This tabular is extracted from the group talk presentation by Dr. Johannes Mohr on 22.April.2016

Abbildung 1.2: LASSO Binary Classification Performance



1.2 Deep Neural Networks

1.2.1 Motivation

In previous sections, we've seen LASSO binary classifier as a solution to sound type classification problem. In the sound type classification module of the Two!Ears system, handcrafted features(rate maps and AMS features) are taken as the input feature vectors of LASSO binary classifiers. In such

simply structured supervised learning model, the quality of features is critical. However, the choice of good features is quite difficult.

Instead of keeping working hard on feature selection, we now introduce deep neural networks(DNNs) to solve the problem. DNNs can learn features with multiple layers and find high level abstractions in data rather than simply using handcrafted features. Because of such excellent characteristic, DNNs are widely used in a variety of machine learning applications. For example, in automatic speech recognition, DNNs are used to train bottleneck features[4]. Also in computer vision, DNNs are excellent at learning a good representation of the visual inputs, which can be used to do semantic image segmentation[5].

1.2.2 DNN Structures

A deep neural network (DNN) is an artificial neural network (ANN) with multiple hidden layers of units between the input and output layers[6][7]. Each layer is made of neurons with learnable weights and biases as parameters of the whole network. Each neuron takes the inner product of outputs from previous layers or input feature vectors with corresponding weights as input and applies an activation function. The common choice of the activation function is the sigmoid function. Fig.1.3 shows a regular deep neural network structure. If all of the neurons in each layer of the network are fully connected to all activation neurons in the previous layer, such a network is called fully connected neural network.

However, such structure is not so efficient with multi-dimensional data like images. Considering a fully connected neural network, the amount of paramters increases with the size of inputs and the number of hidden layers. Too many parameters during training process usually leads to overfitting problem. In this thesis, a variant of regular DNN ,convolutional neural network(CNN), is constructed to solve the problem.

Convolutional Neural Networks

Convolutional Neural Networks (CNN) are biologically-inspired variants of MLPs(Multilayer Perceptron). [8]

The layers of a CNN have neurons arranged in 3 dimensions: width, height, depth. Instead of following the full connection manner, the neurons of layers in CNNs will be only connected to a small region of layer before it. Such region is called receptive field. In this way, the amount of parameters to be learned is decreased compared with fully connected neural networks. Also parameter sharing scheme is used in Convolutional Layers to control

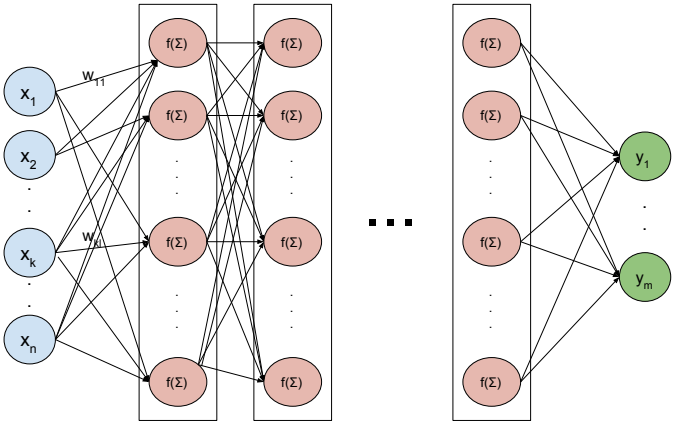


Abbildung 1.3: Deep neural network structure

the number of parameters[9]. In each layer a new data volume is constructed with corresponding layer parameters. At last, an output volume, which is usually one dimensional vector, is constructed as the final output score.

1.2.3 Caffe Toolbox

Caffe[10] is a deep learning framework. It is developed by the Berkeley Vision and Learning Center (BVLC) and by community contributors. Yangqing Jia created the project during his PhD at UC Berkeley. Caffe is released under the BSD 2-Clause license. In this thesis, I used caffe as the toolbox to implement the convolutional neural networks for sound type classification.

Caffe is written in C++ and it also provides the users with command line, Python, and MATLAB interfaces. The command line interface is the caffe tool for model training, scoring, and diagnostics. All training requires a solver configuration file. In this configuration file, the optimization method and its corresponding parameters are defined. To create a Caffe model you need to define the model architecture in a protocol buffer definition file. In solver configuration file, the full path of the model architecture definition file should also be defined.[4]

[4]For further details, please refer to the official website of caffe:http://caffe.berkeleyvision.org/

Caffe provides a variety of layers for users. Generally there are 5 types of layers: vision layers, loss layers, activation/neuron layers, data layers and common layers.

- Vision Layers:
  Convolution, Pooling, Local Response Normalization(LRN),im2col
  Vision layers take images as input and produce new images as output. In this thesis, Convolution and Pooling layers were used.

- Loss Layers:

  Softmax, Sum-of-Squares/Euclidean, Hinge/Margin, Sigmoid Cross-Entropy, Infogain, Accuracy and Top-k
  Loss layers compare an output to a target and assign cost. During the training stage, the loss itself is computed by the forward pass and the gradient w.r.t. the loss is computed by the backward pass. In this way, loss layer drives the learning process by minimizing the loss. In this thesis, Softmax and Sigmoid Cross-Entropy were used.

- Activation/Neuron Layers:
  ReLU / Rectified-Linear and Leaky-ReLU, Sigmoid,TanH / Hyperbolic Tangent,Absolute ,Power,BNLL
  Activation / Neuron layers are element-wise operators, taking one bottom blob and producing one top blob of the same size. In this thesis, ReLU and Sigmoid layers were used.

- Data Layers:
  Database, In-Memory, HDF5 Input, HDF5 Output, Images, Windows, Dummy
  Data enters Caffe through data layers. Use different types of data layers w.r.t different data sources. In this thesis,HDF5Data layers was used, as the handcrafted features, rate maps and AMS features, were written into hdf5 files.

- Common Layers:
  InnerProduct,Splitting,Flattening,Reshape,Concatenation,Slicing, Elementwise Operations,Argmax,Softmax,Mean-Variance Normalization
  Common Layers are implemented for simple data operations. In this thesis, InnerProduct, Concatenation,Slicing and Sigmoid layers were used.

1.3 Aims of Thesis

In my thesis, I investigate different architectures of DNNs and different loss functions to improve the performance of this sound type classification system. I also tried to use data augmentation to overcome the overfitting problem.

Kapitel 2

Comparison of Architectures and Loss Functions

In this chapter, I will first introduce some basic types of layers applied in the construction of the DNNs in this thesis. Then I will describe three DNN architectures for Sound Type Classification, One-Against-All model, SoftmaxWithLoss model and SigmoidCrossEntropy model. At last, I will show the experiment results of these three architectures.

2.1 Methods

In order to accelerate the training process, GPU mode was prefered as the solver_mode in all of the experiments.

The performance metric is sensitivity, specificity and balanced accuracy. For a classifier, which classifies all the test sources as positive, it still get a 100% sensitivity. If the amount of true positive points is equal to the amount of true negative points, the accuracy can still reach 50%. Considering only the accuracy, the classifier tends to make positive or negative decisions w.r.t the prior of positive and negative sources. That is why sensitivity, specificity and balanced accuracy were chosen as the performance metrics. Caffe didn't implement a layer for the calculation of sensitivity and specificity. The Accuracy layer produces a metric which can't describe the performance of a classifier well. In order to calculate the sensitivity, I used the python interface in Caffe to extract intermediate outputs of the DNN from the trained model files and calculate these three performance metrics to complete the comparison of different DNN architectures. There are in total 11 classes of sound types, so the performance metrics were calculated for each of these sound types.

2.1.1 Layer Types for Sound Type Classification

2.1.2 Data Layer

As feature representation for sound type classification, rate maps and AMS features enter the network through data layers. Since both the training dataset and the test dataset are stored in HDF5 files, the data layer type used in this thesis is "HDF5Datalayer.

Convolutional Layer

Convolutional layers produce intermediate feature images. As discussed in previous chapter. The adoption of convolutional layers helps in decreasing the weights, which has a positive influence in preventing the overfitting problem.

Pooling Layer

Pooling layers can be seen as a downsampling operation. The input features are convoluted with simple filters, which is usually of size 3×3. If the filter operation is to select the maximum value in the neighborhood, such Pooling Layers are called Max Pooling Layers. Pooling layers add local translation invariance to the network, which improves the generalization of the network[11]. Pooling layers also reduce the amount of parameters and computation in the network, and hence also control the overfitting problem.

ReLU Layer

ReLU layer implements the activation function, defined as:

$$f(x) = max(0, x) \tag{2.1}$$

In deep neural networks, back propagation through multiple layers with sigmoid activation function may cause the gradient vanish. Compared with sigmoid activation function, ReLU has a constant gradient, as a result, it can be used to reduce likelihood of vanishing gradient. That's why ReLU layers are used in our DNN model.[12]

Dropout Layer

Dropout layers randomly switch off part of neurons in the network during training process and increase the output values for the other neurons to normalize. As a result, Dropout layers also help to control overfitting. Also it

provides a way of implicitly combining many different neural network architectures efficiently[13]. The percentage of dropped neurons can be defined as a parameter of dropout layer in Caffe. In this thesis, this parameter was set to 0.5.

2.1.3 Architectures and Loss Functions

In this section, three types of DNN architectures for Sound Type Classification will be introduced. Each of these three architecture models are constructed from the layers mentioned in previous section. Also two types of loss functions will be discussed in this section together with corresponding architectures.

One-Against-All Model

Sound type classification can be seen as a multi-class classification problem. If we don't consider multi-label classification, a simple model which gives out one scalar output that indicates the class label can be easily constructed in DNN. This model is called One-Against-All model in this thesis. One-Against-All model trains a classifier for 12 classes(11 named classes and a general 'anything else' class). Components of this architecture are shown in Fig.2.1. The numbers annotated next to the arrows denote the dimensionalities of output vectors from previous layers. One-Against-All model consist of 2 Convolutional layers, 3 ReLU layers, 1 Maxpooling layer, 2 Innerproduct layer,1 Dropout layer and 1 SoftmaxWithLoss Layer. The first Convolutional layer has a kernel size of 8×6 and the second Convolutional layer has a kernel size of 3×3. The output of the second Innerproduct layer is a 12-dimensional vector. The loss layer used in One-Against-All model is SoftmaxWithloss layer. The SoftmaxWithLoss layer computes the multinomial logistic loss of the softmax of its input scores. It takes a batch size of score vectors and true label vectors as inputs. The input scores from previous Innerproduct layer indicates the probability of the assignment to one class label, in other words, the output label would be the index with largest score value in the score vector.

$$\hat{p}_{nk} = exp(x_{nk})/[\Sigma_{k'} exp(x_{nk'})] \tag{2.2}$$

$$E = -\frac{1}{N}\Sigma_{n=1}^{N} log(\hat{p}_{nl_n}) \tag{2.3}$$

Where $\hat{p}_{nk}$ means the probability of input source indexed as n to be classified as class k.
x_{nk} is the input score of the loss layer. n indicates the input source, k is

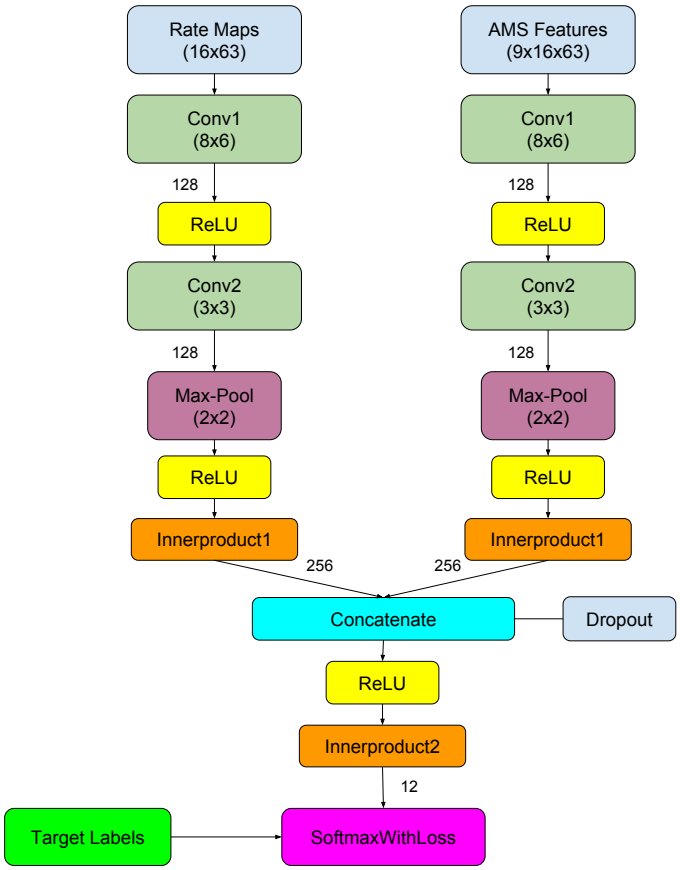


Abbildung 2.1: One-Against-All Model architecture visualization

the index of the score vector from previous layer. The score vector is a 12-dimensional including the score for 'anything else' class.
E is the computed cross-entropy classification loss. l_n is the target label of input source indexed as n.

SoftmaxWithLoss Model

If we consider to build 11 binary classifiers inside one DNN, based on the idea of LASSO binary classifier, the new model can produce a label vector. As a result, this new DNN model can also do the multi-label classification. In this thesis we call this new model SoftmaxWithLoss model. The architecture is shown in Fig.2.1.3. The basic architecture of the SoftmaxWithLoss model is almost the same as the One-Against-All model. In order to construct a classifier bank inside this network, some changes on the layer that outputs the score vectors should be made. In One-Against-All model, the scores come

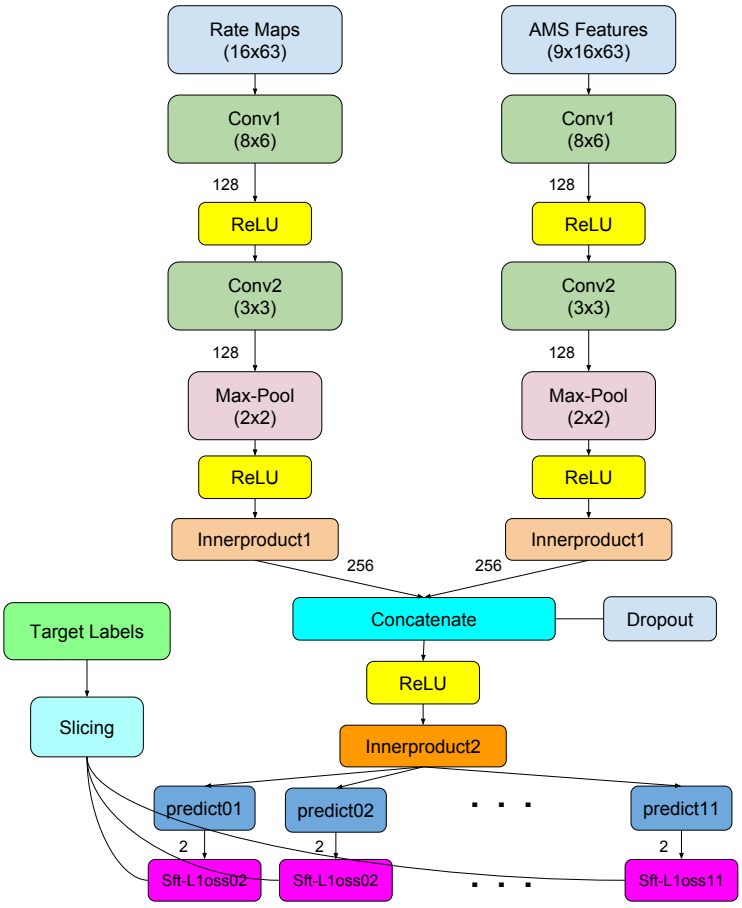


Abbildung 2.2: SoftmaxWithLoss Model architecture visualization

from an Innerproduct layer, and were passed directly to SoftmaxWithLoss Layer, which calculate an overall loss value as the object function for back propagation. In this SoftmaxWithLoss Model, instead of constructing a 12-dimensional score vector, the output from previous Innerproduct layer, which is in this case a 256-dimensional vector, are further sent to 11 independent Innerproduct layers, the new scores become 2-dimensional vectors and go to 11 independent SoftmaxWithloss layers. The connection between these Innerproduct layers and SoftmaxWithloss layers are one-to-one. At this point, the network is split into 11 branches to train 11 binary classifiers. In each iteration of the model optimization, the model takes a batch size of input sources and split the true labels of the sources into 11 groups w.r.t the label values. The split of true labels are carried out by one Slice layer. These 11 groups of true labels are then passed to these 11 SoftmaxWithLoss layers se-

parately. During the back propagation, the parameters of these Innerproduct layers will be trained to achieve 11 binary classification tasks.

SigmoidCrossEntropy Model

The application of SoftmaxWithLoss layer makes the classification decision in a probabilistic way. If we want to use it for multi-label classification, the only way is to construct a bank of binary classifiers as the SoftmaxWithLoss model. If we consider another decision policy, that sets threshold for the scores and gives out a binary label vector, this also solves the multi-label classification problem. Sigmoid function can maps the scores to probability prediction vectors, each of which only answer the yes-or-no question w.r.t a decided threshold. Inspired by this idea, the SigmoidCrossEntropy Model is constructed using SigmoidCrossEntropy as the loss function.

$$\hat{p}_n = \sigma(x_n) \tag{2.4}$$

$$E = -1\frac{1}{n}\Sigma_{n=1}^{N}[y_n log\hat{p}_n + (1-y_n)log(1-\hat{p}_n)] \tag{2.5}$$

Where $\hat{p}_n$ is the mapped probability from score x_n.
y_n is the target label of source index as n. E is the computed cross-entropy classification loss. The architecture components are almost the same as those in One-Against-All model. Instead of using SoftmaxWithLoss as the loss layer, SigmoidCrossEntropy layer is applied in this model. The architecture visualization is shown in Fig.2.1.3

2.1.4 Implementation Details

The training dataset contains 417888 feature vectors. For a batch size of 128, it requires 3265 iterations to go through all the training data. So I set the maximum iteration to 100000 to make sure it converges. The optimization method used in this chapter is Stochastic Gradient Descent(SGD). The initialization strategy for the SGD solvers was the strategy used by Krizhevsky et al. [5] in their famously winning CNN entry to the ILSVRC-2012 competition, which has been proved to be a good strategy. The general solver protocol settings are listed in Tab.2.1.4 I also experimented a bit with the

base_lr	lr_policy	gamma	stepsize	momentum	weight_decay	solver_mode
1e-4	step	0.1	30000	0.99	0.0005	GPU

Tabelle 2.1: Optimization Solver Settings

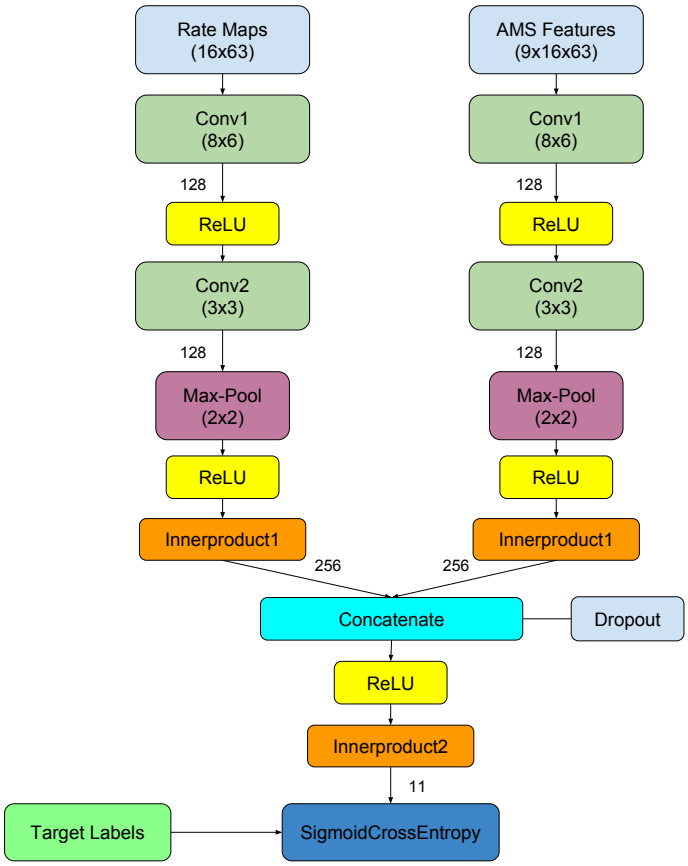


Abbildung 2.3: SigmoidCrossEntropy Model architecture visualization

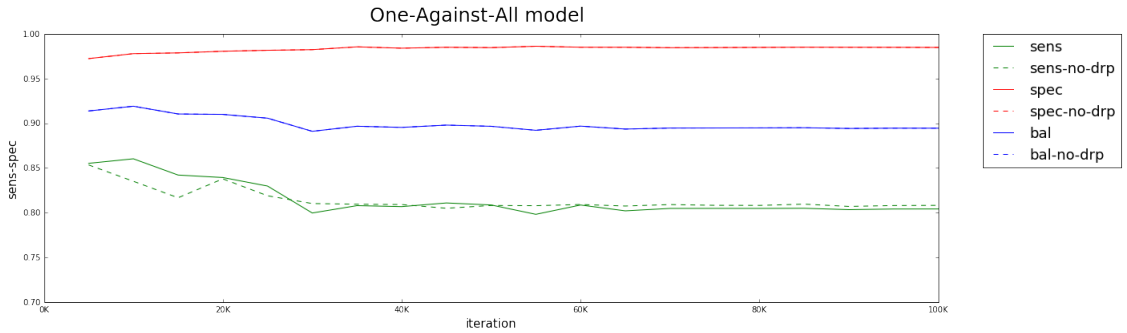
Dropout Layer. For each of these three models, I carried out 2 experiments, one with dropout layer, the other without. The dropout ratio were all set to 0.5.

2.2 Results and Discussion

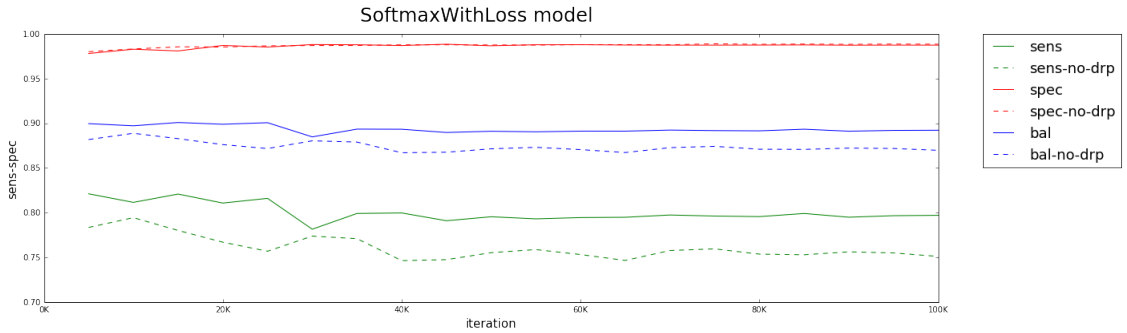
2.2.1 Learning Curve

The learning curve of three architectures, displaying the performance metrics during 100000 iterations, are shown in Fig.2.4. In each diagram, the solid lines represent the results of the model with a dropout layer while the dash line without dropout layer. In the legend, 'sens' means sensitivity, 'spec' means specificity and 'bal' means the balanced accuracy. The learning curve shows that, all of these three models converged before 100000 iterations. Also for

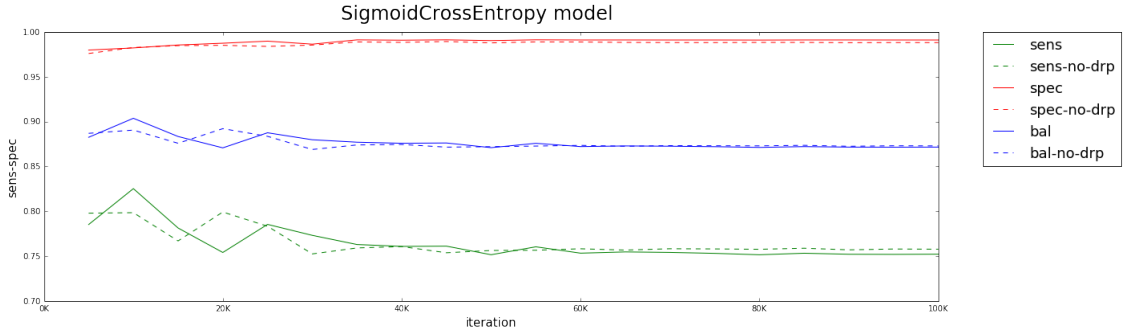
One-Against-All model and SigmoidCrossEntropy model, the difference of the performances seems not so obvious. While in SoftmaxWithLoss model, the adoption of dropout layer contributes a lot for the good performance. All of these three models get quite high specificity while quite low sensitivity. This means these three models tend to make negative decision. In the rest of this thesis, the main task is to tackle this problem, in order to find a better model with satisfying sensitivity.



(a)



(b)



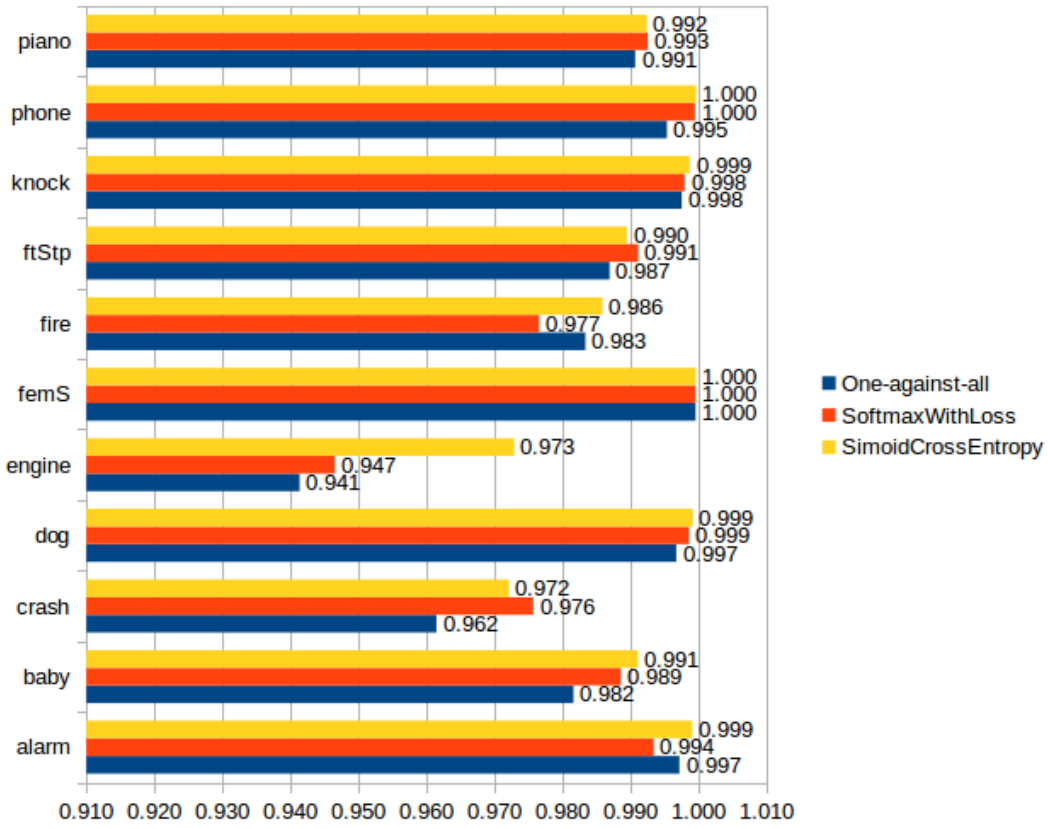
(c)

Abbildung 2.4: Learning curve of 3 architectures

2.2.2 Comparison of Three Architectures

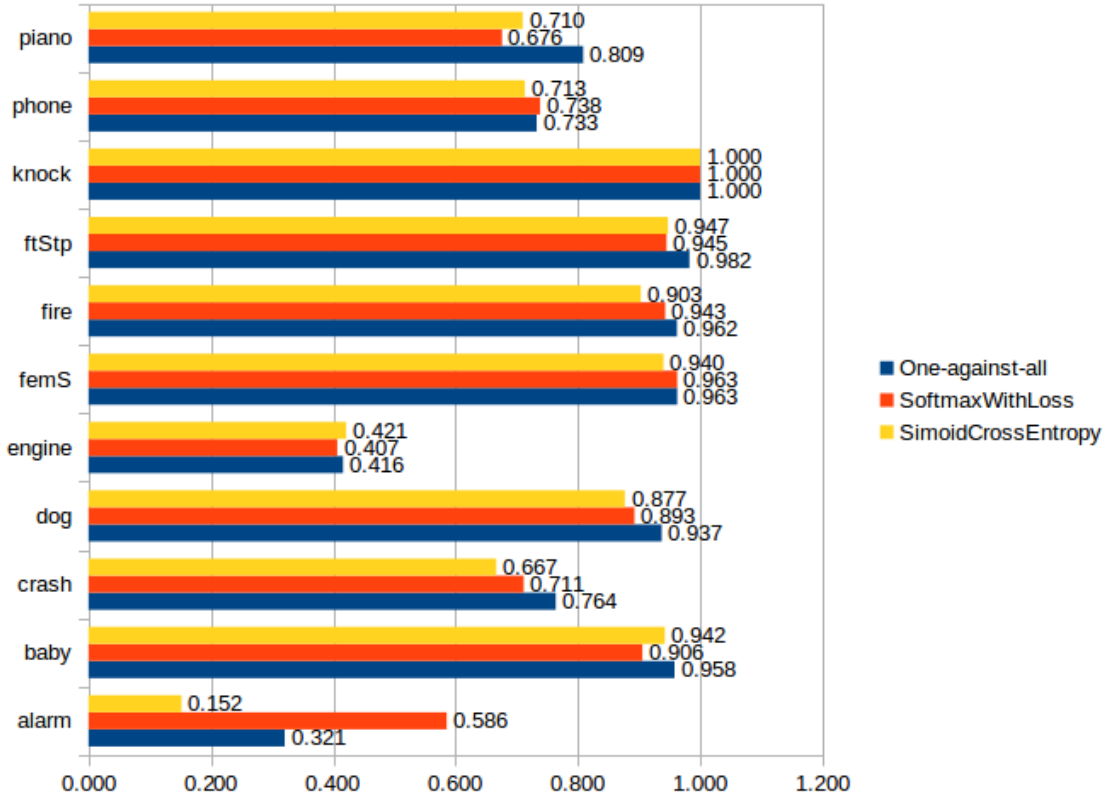
Fig.2.5 shows the specificity accuracy metric for these three architectures. As mentioned above, there is not much difference when comparing these three models with sensitivity metric. However it shows much difference when focusing on the sensitivity metric. As shown in Fig.2.6, the One-Against-All model shows a pretty good performance. It seems that SigmoidCrossEntropy model does not perform well. Considering the decision policy, SigmoidCrossEntropy model requires a proper setting of threshold. To see how well SigmoidCrossEntropy model can perform, I also plotted the Receiver Operating Characteristic(ROC) curve, see Fig.2.7. When classifying test sources from type 'alarm', 'crash' and 'engine', the performance has more critical limits than the others. For the other 8 sound types, SigmoidCrossEntropy can still get a good performance if the threshold is properly set. From the sensitivity bar chart Fig.2.6, 'phone' and 'piano' are also considered as sound types that are difficult to make a classification decision for SoftmaxWithLoss model and One-Against-All model. So in the rest of this thesis, I will focus on SigoidCrossentropy model and do some other improvement based on this model.

Abbildung 2.5: Specificity of 3 models(with dropout layer)



To check whether such bad performance comes from the overfitting problem. I also carried out an experiment for SigmoidCrossEntropy model, where I replaced the test dataset with training dataset. The ROC curve of this experiment is shown in Fig.2.8. When testing on the training dataset, SigmoidCrossEntropy model shows a nearly perfect performance. This means the current SigmoidCrossEntropy model meets the overfitting problem. In the rest of this thesis, I will mainly focus on how to deal with this overfitting problem.

Abbildung 2.6: Sensitivity of 3 models(with dropout layer)



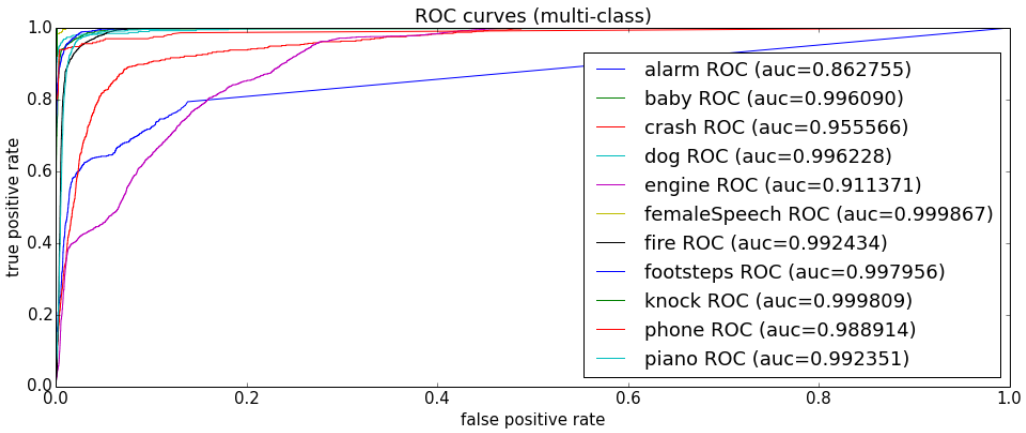
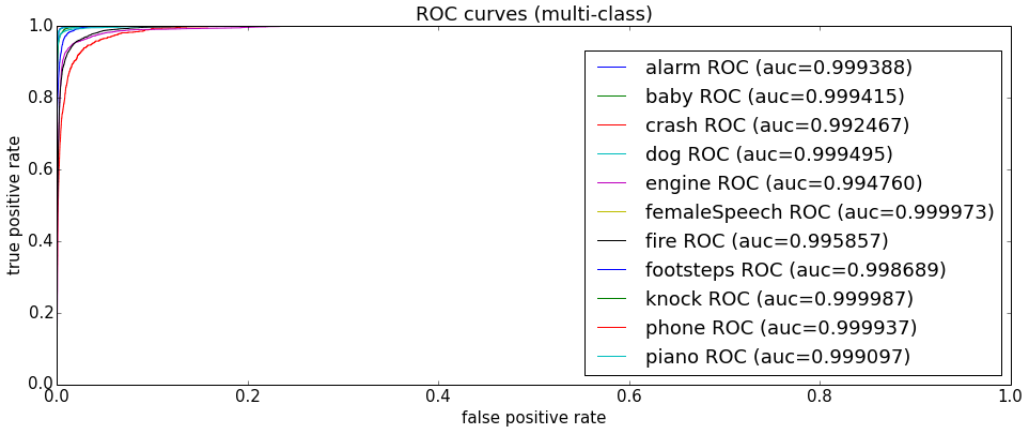


Abbildung 2.7: ROC curve of SigmoidCrossEntropy model(with dropout layer)

Abbildung 2.8: SigmoidCrossEntropy model tested on training dataset



Kapitel 3

Data Augmentation

In this Chapter, data augmentation is introduced to control the overfitting problem. For sound type classification, there are mainly two policies for data augmentation. The effects of different data augmentation policies will also be discussed in this chapter.

3.1 Overview and Motivation

The experiment results in previous chapter show that the training of SigmoidCrossEntropy model meets the overfitting problem. Overfitting usually comes from over trained parameters. If we consider this problem in another way, that is if we have adequate training data to abstract the high level data structure, it may control the overfitting problem to a certain extent.

Data augmentation was adopted by Krizhevsky et al. to combat the overfitting problem for ImageNet Classification.[5] The easiest and most common method to reduce overfitting on image data is to artificially enlarge the dataset using label-preserving transformations."

What data augmentation does in my experiments is to randomly shift the rate maps and AMS feature over time and frequency domain. Intuitively data augmentation increase the irrelevant variability of training data, which helps the model to find a high level abstraction of the data. Shifting over time domain only and shifting over both time and frequency domain can be seen as two shifting policies.

However, such operation is not implemented in Caffe. In the meanwhile, Caffe provides a special type of layer, Python layer, which enables Caffe users to implement self-defined layers in python. I implemented a Python layer for the augmentation operation. The missed data values due to shifting operation are replaced by normal distributed vectors with mean and covariance matrix

calculated from the original input features. Instead of padding with constant values, padding with normal distributed vectors add noise in the data rather than changing the data pattern. Such noise adds the classification irrelevant variability in the training data and hence may control the overfitting problem. The shifting step size is randomly generated in each iteration. I also set a range for the shifting steps, this range can be defined by users through the layer parameters. I experimented on a series of this range parameters.

In the following sections, the parameter settings will be described in form $T(minT, maxT), F(minF, maxF)$. T means shifting over time domain. The shifting step in each forward propagation iteration is a integer, which is uniformly distributed in range $(minT, maxT)$. One shifting step means shifting by one time frame. Positive step means shifting left, while negative step means shifting right. F means shifting over frequency domain. The same as shifting over time domain, the shifting step in each forward propagation iteration over frequency domain is also a integer, which is uniformly distributed in range $(minF, maxF)$. One shifting step over frequency domain means shifting by one frequency channel. Positive step means shifting to higher frequency channels, while negative step means shifting to lower frequency channels.

3.2 Methods

To compare the performance statistically, I carried out dependent t-tests using the 11-dimensional statistics(performance metrics) between two experiments. T-tests are widely used to determine if two sets of data are significantly different from each other. Large p-value, e.g. greater than 0.05 or 0.1, rejects the null hypothesis of identical average scores. Small p-value smaller than the threshold, e.g. 0.01, 0.05,0.1, rejects the null hypothesis of equal averages. In this way, the comparison of performance between two experiments can be seen from this statistic, p-value.

3.3 Results and Discussion

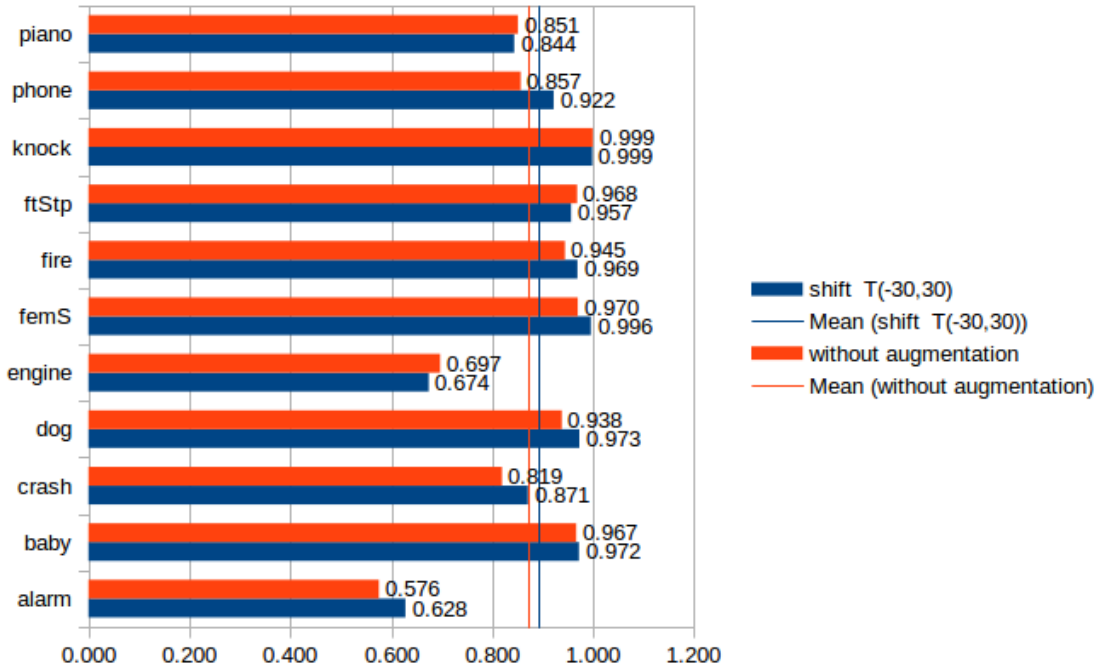
In this section I will mainly compare the experiment results in three steps. Firstly I will compare the new model with data augmentation with the old model in previous chapter. Secondly I will compare 2 shifting policies, shifting over time domain and shifting over both time and frequency domain. At last I will compare different settings of the shifting range parameters used in the Python layer to roughly get a best setting of parameters.

3.3.1 Data Augmentation effects

In the experiment with data augmentation, I set the shifting range to $T(-30, 30)$. The other experiment result to compared with is the SigmoidCrossEntropy model from Chapter 2.

The experiment results are shown in Fig.3.1. Two vertical lines represent for the mean value lines for these two experiments. The model with data augmentation outperforms the SigmoidCrossEntropy model for most sound type classes. I also carried out a dependent t-test with the balanced accuracy statistics from these two experiments, the p-value I got is $0.049577 < 0.05$ which means the performance statistics of two models are indeed significantly different from each other. Data augmentation improved the model performance.

Abbildung 3.1: Balanced accuracy of two models, shift over T(-30,30)

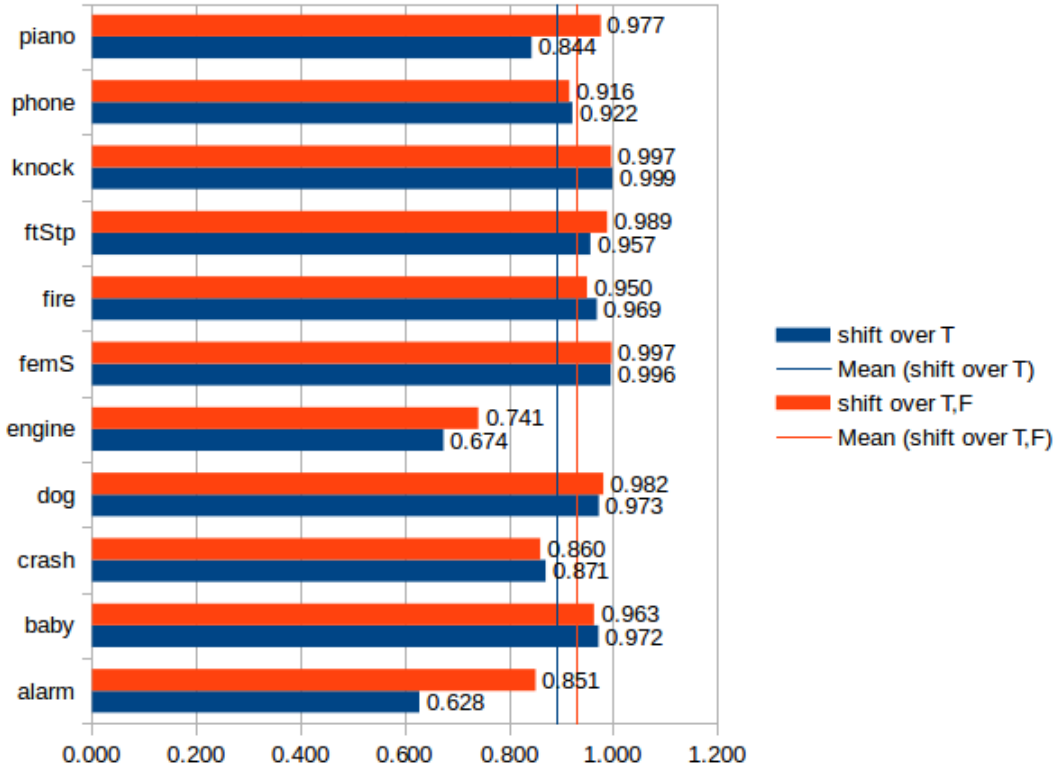


3.3.2 Shifting policies

Besides shifting over time domain, shifting over both frequency domain and time domain is another shifting policy. Shifting over time domain adds the variability of recording delay. Shifting over frequency domain adds the variability of basic frequency of the sound source. These two variability can

be both considered as irrelevant. The parameter setting for shifting over

Abbildung 3.2: Balanced accuracy of two shifting policies,shift over T(-30,30),shift over T(-30,30),F(-8,8)



time domain policy is the same as that described in section 3.3.1, which is $T(-30, 30)$. The parameter setting for shifting over time and frequency domain is $T(-30, 30), F(-8, 8)$. For 6 of the sound types, shifting over time and frequency domain outperforms shifting over time domain. The averaged balanced accuracy over 11 sound types of shifting over time and frequency domain is greater than the other shifting policy. So in the following experiments of shifting parameters, I preferred to adopt the shifting over time and frequency policy.

3.3.3 Shifting Parameters

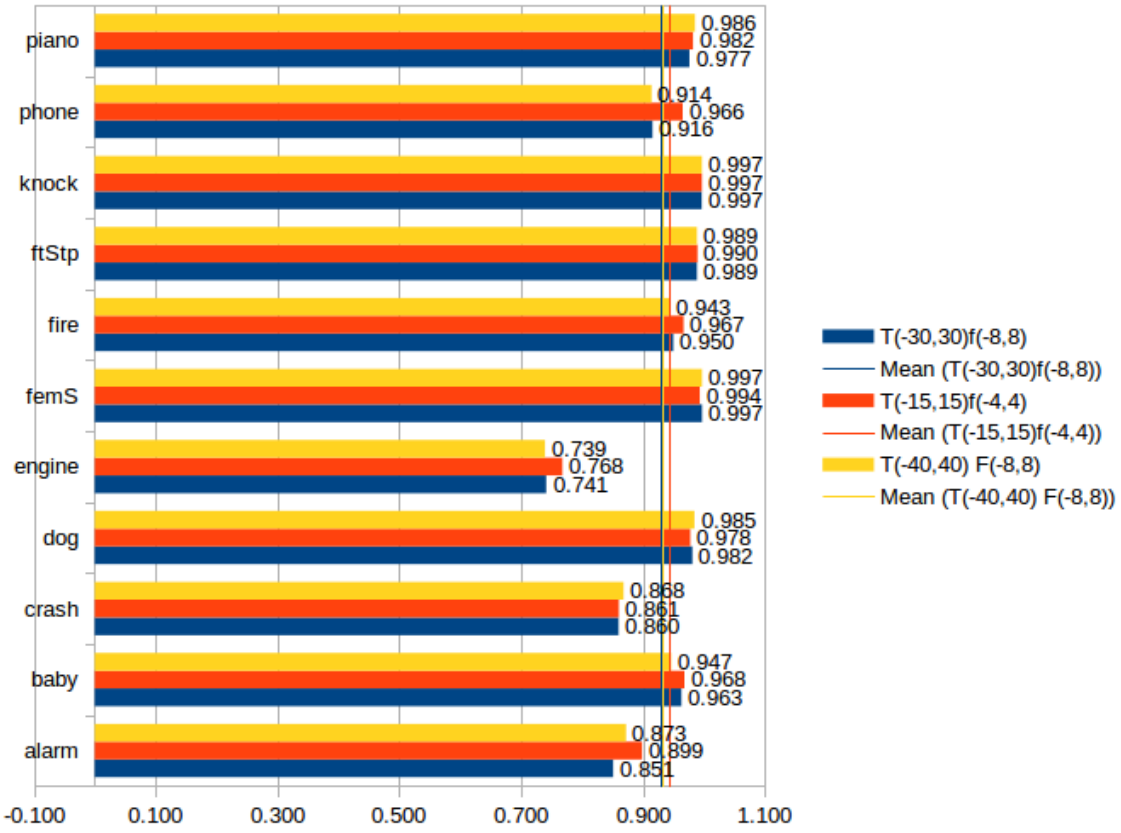
To roughly find a good parameter settings for data augmentation, I carried out three experiments with different parameter settings:

- $T(-30, 30), F(-8, 8)$

- $T(-15, 15), F(-4, 4)$
- $T(-40, 40), F(-8, 8)$

Fig.3.3 shows the balanced accuracy and Fig.3.4 shows the sensitivity metric of these three experiments. From the vertical lines in the diagram, which

Abbildung 3.3: Balanced accuracy of 3 parameter settings

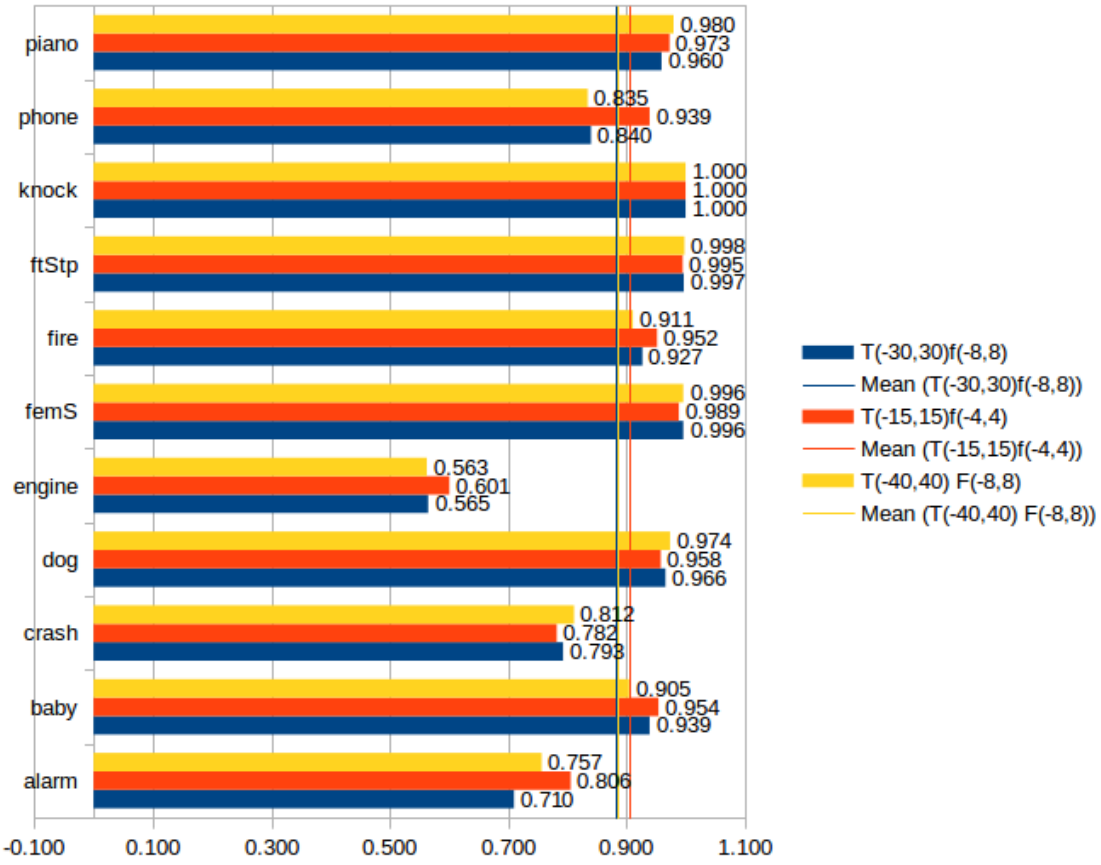


indicates the mean performance metrics over sound type classes, we can see that the averaged balanced accuracy and the averaged sensitivity of parameter setting $T(-15, 15), F(-4, 4)$ are the highest among all of these three parameter settings. So currently the advisable shifting parameters would be $T(-15, 15)F(-4, 4)$.

3.3.4 Summarize and Conclusion

The averaged balanced accuracy over 11 sound type classes of the experiments in this chapter are listed in Tab.3.3.4. The performance metric values

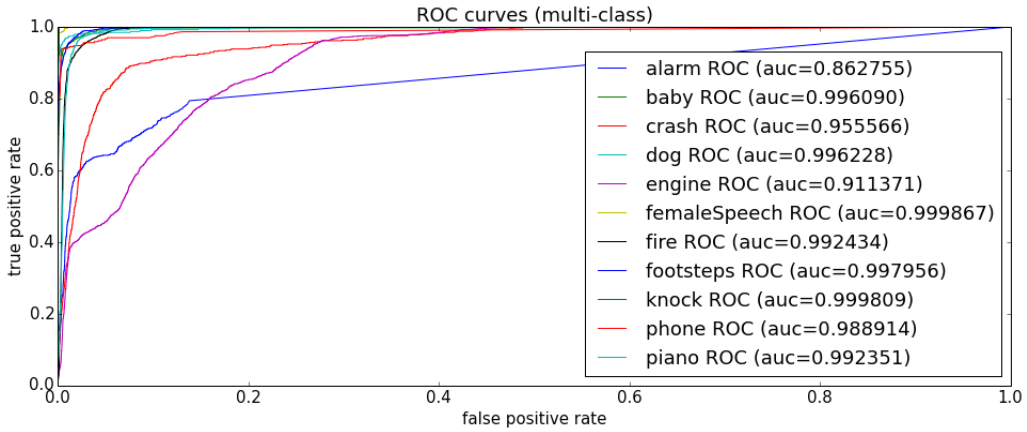
Abbildung 3.4: Sensitivity of 3 parameter settings



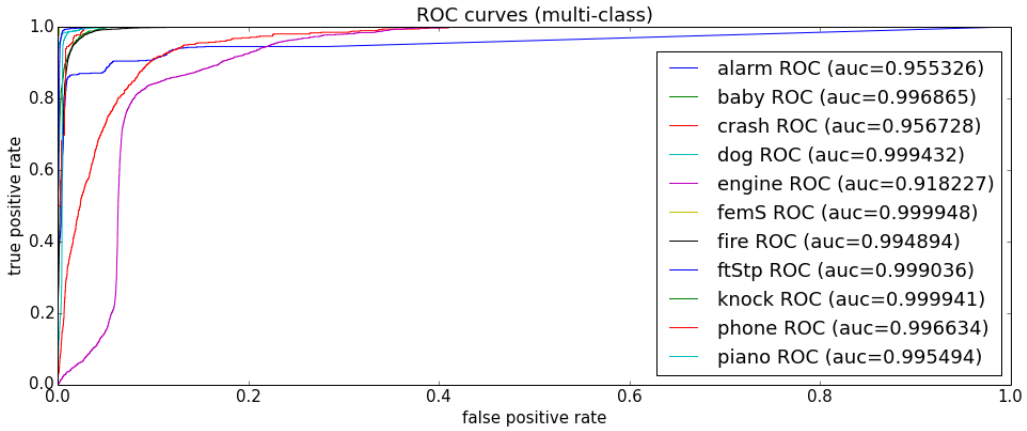
in bold are the best values among these 5 experiments. The best balanced accuracy is acquired in the experiment with data augmentation with parameter setting $T(-15, 15)F(-4, 4)$. Also if we compare the ROC curves of SigmoidCrossEntropy model from previous chapter and the model with data augmentation constructed in this chapter, we can see that the 'auc'(area under curve) values of SigmoidCrossEntropy model with data augmentation for all of these 11 sound type classes are over 0.95, while the 'auc' values of SigmoidCrossEntropy model without data augmentation for sound type 'alarm' and 'engine' are below 0.95. Thus we can conclude that data augmentation can be adopted as a method to control overfitting problem. The ROC curves are shown in Fig.3.5

	balanced accuracy	sensitivity	specificity
no data augmentation	0.872	0.752	**0.991**
T(-30,30)	0.891	0.799	0.984
T(-30,30),F(-8,8)	0.929	0.881	0.978
T(-15,15)F(-4,4)	**0.943**	**0.905**	0.981
T(-40,40)F(-4,4)	0.931	0.885	0.977

Tabelle 3.1: Averaged performance metrics for 5 experiments



(a) SigmoidCrossEntropy model without data augmentation



(b) SigmoidCrossEntropy model with data augmentation $T(-15, 15), F(-4, 4)$

Abbildung 3.5: ROC Curves for models with and without Data Augmentation

Kapitel 4

Conclusions and Outlook

In this thesis, I first constructed three DNN architectures and focused on SigmoidCrossEntropy due to its advantage in extension of multi-label classification and its good receiver operating characteristic as shown in Fig.3.5 in section 3.3.4. The application of data augmentation improves the averaged balanced accuracy by 7.1%(from 87.2% to 94.3%). In conclusion, data augmentation helps in controlling overfitting problem.

As shown in Chapter3, the experiment results are sensitive to parameter settings. In future work, finding a optimal parameters setting could be one research direction.

Besides data augmentation, there are more options to conquer the overfitting problem, such as adding more dropout layers into the network, altering the kernel size for convolutional layers to decrease the amount of weights in the network. Also good parameters tuning would help a lot when optimizing the DNN architecture. However such optimization is computationally very costly. A full learning process of a DNN model goes through the entire data set multiple times. Each alteration of a hyper parameter requires such a lengthy learning process.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of machine learning*. MIT press, 2012.

[2] N. Ma, I. Trowitzsch, J. Taghia, C. Schymura, D. Kolossa, T. Walther, H. Wierstorf, T. May, and G. Brown, "Software architecture two!ears deliverable 3.2," tech. rep., Two!Ears Project Group, 2014.

[3] T. May and T. Dau, "Computational speech segregation based on an auditory-inspired modulation analysis," *The Journal of the Acoustical Society of America*, vol. 136, no. 6, pp. 3350–3359, 2014.

[4] Y. Zhang, E. Chuangsuwanich, and J. R. Glass, "Extracting deep neural network bottleneck features using low-rank matrix factorization.," in *ICASSP*, pp. 185–189, 2014.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[6] B. Yoshua, "Learning deep architectures for ai," *Foundations and trends in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[7] S. Jurgen, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

[8] D. . L. Lab, "Convolutional neural networks (lenet) deeplearning 0.1 documentation," August 2013.

[9] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[10] J. Yangqing, S. Evan, D. Jeff, K. Sergey, L. Jonathan, G. Ross, G. Sergio, and D. Trevor, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[11] 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.

[12] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pp. 807–814, 2010.

[13] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting.," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[14] V. Vapnik, *The nature of statistical learning theory*. Springer Science & Business Media, 2013.