Audio Event Recognition

Zhenjia Xu
ACM Honored Class
Shanghai Jiao Tong University
xuzhenjia@sjtu.edu.cn
515030910637

Abstract

Deep learning has recently achieved great success on various tasks, especially on image classification. One of the well known models, ResNet[3], has achieved 3.57% error on the ImageNet[1] test set, which even beats human-level performance of 5%. Audio event recognition ability is also of great significance to human. In this project, I tend to use deep learning to recognize audio on the Audio Set[2]. At the same time, I made a comparison of different frameworks used in deep learning, including Tensorflow, Pytorch and Mxnet. Finally, a large amount of experiments related to deep learning are conducted, such as performance of different architecture, how to prevent overfitting and effects of hyperparameters.

1 Introduction

1.1 Background

Now that AI has the ability to classify images, it should also learn to recognize audio events. Sounds carry a large amount of information about our everyday environment and physical events that take place in it. Humans are very skilled in perceiving the general characteristics of the sound scene around them, whether it is a busy street, a quiet park or a quiet office environment, and recognizing individual sound sources in the scenes, such as cars passing by, birds, or footsteps.

1.2 Dataset

Audio Set is used in this project. It consists of an expanding ontology of 632 audio event classes and a collection of 2,084,320 human-labeled 10-second sound clips drawn from YouTube videos. However, instead of using raw audios, features of 128 dimensions, extracted from a bottlenect layer of ResNet, are actually used in the project.

2 Task

Notation in this paper are displayed in Table 1. The task of this project is audio event recogtion. More Specifically, given the feature of an utterance, the goal is to predict the event according to the utterance. Note that an utterance may correspond to several events, which means this task can be classified as multicalss classification. In this problem, we aim to minimize the loss function L which measures the prediction made by the model. Formally, suppose we have a model M parameterized by θ, the trainning process can be considered as an optimization problem:

$$\textbf{minimize} \quad L(M_\theta, X_{train}, y_{train})$$

where the input of the model M is $X \in \mathbb{R}^{N*1280}$, the output is $pred = M_\theta(X) \in \mathbb{R}^{N*527}$ represents the probability of each events eppearing in the utterance, and the loss function L we used is cross entropy.

$$L(pred, y) = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{527} y_{i,j} * \log(pred_{i,j}) + (1 - y_{i,j}) * \log(1 - pred_{i,j})$$

Two metrics atr used to evaluate the performance the model.

- **AUC**
  The area under the ROC convex hull.
- **AP** The area under presicion-recall curve.

Table 1: Notations and descriptions

Notation	Description
X_{train}, y_{train}	The trainning dataset splitted from the origin dataset.
X_{eval}, y_{eval}	The validation dataset splitted from the origin dataset.
X_{test}, y_{test}	The testing dataset.
$X_{i,j}$	the j^{th} feature of $data_i$.
$y_{i,j}$	Whether $data_i$ contains $event_j$. $(y_{i,j} \in \{0, 1\})$
$pred_{i,j}$	The probability of $event_j$ appearing in $data_i$ predicted by the model. $(pred_{i,j} \in [0, 1])$

3 Experiments

3.1 Flexible API

In order to support various deep learning framework such as Tensorflow, Pytorch, Mxnet, I designed a flexible API to build neural network of different structure. The description of a deep learning model consists three main elements: structure, loss function and optimizer.

- **Structure**
  The structure contains the detailed information of each layer. This API supports three types of layer: fully connected layer, residual block, convolution layer. Figure 1 demonstrates the example of these layers.

```
get_hparam(
    type='fc',
    units=256,
    activation='relu',
    stddev=1e-3,
    dropout=0.5,
    batchnorm=True,
),
```

(a) fully-connected

```
get_hparam(
    type='res',
    units=[512, 512],
    stddev=1e-3,
    activation='relu',
    dropout=0.5,
    batchnorm=True,
),
```

(b) residual

```
get_hparam(
    type='conv',
    filters=64,
    kernel_size=[3,7],
    strides=[1, 2],
    activation='relu',
),
```

(c) concolution

Figure 1: Examples of three layers

- **Loss Function**
  There are two loss functions supported in this API
  - mean squared error
  - sigmoid cross entropy
- **Optimizer**
  In this API, three frequently used optimizer, Adam, RMSProp and SGD-momentum are implemented. Learning rate is one of the most important hyper parameter.

This API plays the role of an intermediate representation of neural network. The backend if just translate the description into different deep learning framework. Detail about the API refers to the "/Project/Model"

3.2 Comparison of Different Frameworks

The experiments mentioned later were implemented by three popular frameworks in deep learning: Tensorflow, Pytorch, Mxnet. I made a comparison base on my personal experience(Table 2). The tests of performance and speed are based on the ResNet model, with the same structure, optimizer and initializer (Hyperparameters of the optimizer may be different). From my persepective, I prefer Pytorch compared with the other two frameworks.

	Tensorflow	Pytorch	Mxnet
Difficulty to get started	difficult	easy	easy
API	good	medium	bad
Performance(AUC)	0.939	0.941	0.938
Performance(AP)	0.198	0.203	0.198
Speed	2m33s	2m13s	2m14s

Table 2: Comparison of different frameworks

3.3 Comparison of Different Network Architectures

Three popular architecutres were implemented. They are simple fully-connected network, residual network and convolution network. The performance of these three architectures are shown in Table 3. I also tried the simplest ensemble method, just average the score of the three models. It's a little bit surprising that the fully-connected network achieved the best performance. The reason may be because the feature extracted by the ResNet is very clean and all we neeed to do is just a simple classification task. The resudual block is useful for deep network, and the convolution layer is good at extracting features which has been done before. As a result, although the fully-connected network is simple, it fits the task best.

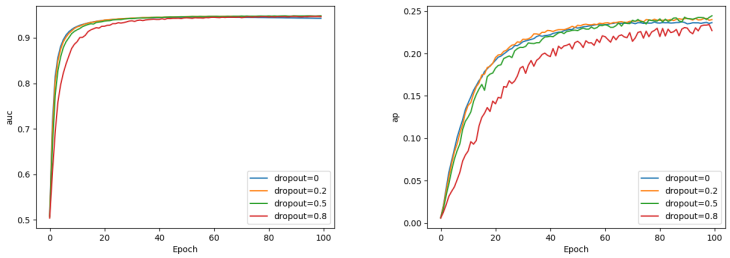
	fully-connected	residual	convolution	ensemble
AUC	0.942	0.941	0.926	0.949
AP	0.220	0.203	0.207	0.258

Table 3: Performance of different architecture

3.4 Fight with Overfitting

Due to the small size of data, the phenomenon of overfitting is vary serious, especially when the network is deep and wide. I tried three common ways to fight with overfitting.

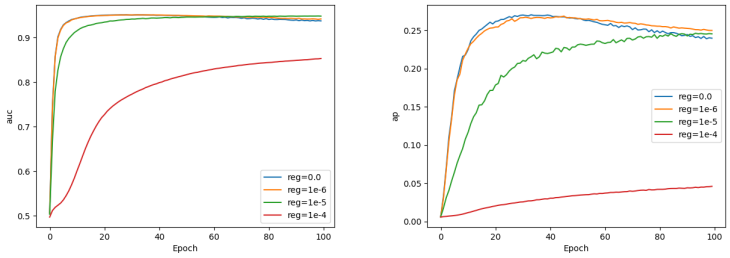
- **Dropout**
  Different dropout rates have been tried. According to Figure 2, we can see that dropout has almost no effect in this project.



(a) AUC - dropout (b) AP - dropout

Figure 2: The effect of different dropout rate

- **Regularization**
  Another way to solve overfitting is using regularization loss. Training curve (Figure 3) shows that proper regularization hyper parameter is of vital importance, not only preventing overfitting, but also accelerating converge.

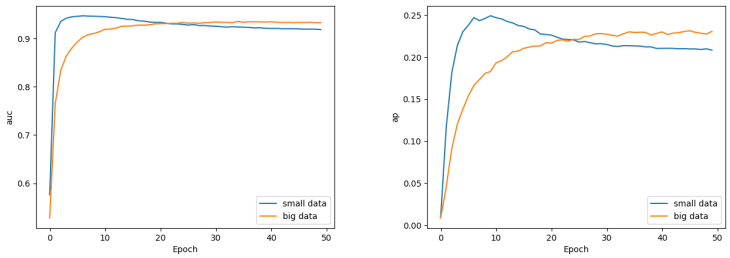


(a) AUC - regularization (b) AP - regularization

Figure 3: The effect of different regularization rate

- **Big data**
  There is no doubt that the thorogh solution to overfitting is using big data to train the model. The experiment mentioned before used small data, the size of which is about $20K$. I also tried big data (size $\approx 2M$) to train models. Figure 4 and Table 4 show the performance of big data, compared to with

small data. We can see that using big data can prevent overfitting to a large degree, but the speed of convergence is been slowed down.



(a) AUC - big data (b) AP - big data

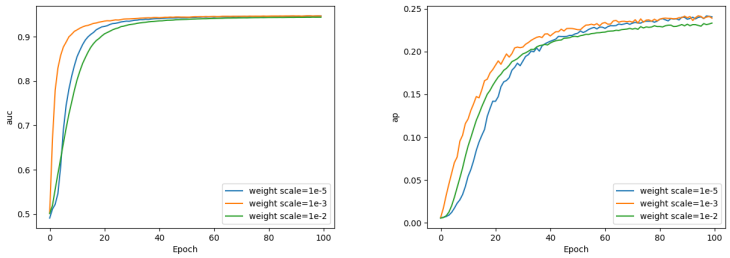
Figure 4: The effect of using big data

	fully-connected	residual
AUC	0.944	0.942
AP	0.232	0.210

Table 4: Performance of big data

3.5 Some Other Experiments

I also studied the impact of weight scale and width, whose trainning curves are shown in Figure 5 and Figure 6 respectively. According to these curves, we can draw the following conclusions: In this project, weight scale of initialization and the width of each layer are not much related to the final performance, but has a great influence to the speed of convergence. A proper weight scale can and width can help speed up the convergence of model to a large extent. The reason for this phenomenon may be because the model used in the experiment is a two-layer vanilla fully-connected network.

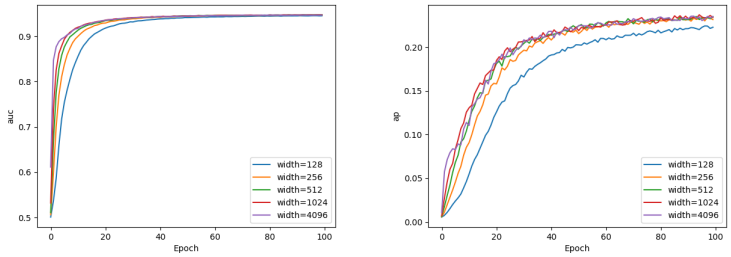


(a) AUC - weightscale (b) AP - weightscale

Figure 5: The effect of different weightscale

4 Conclusion

To sum up, I firstly design a flexible API to support different baskend deep learning frameworks. Then, I made an attempt to use different frameworks and network architectures to improve the performance. At the same time, I make an comparison of these frameworks and architectures. Finally, a large variety of experiments releted to deep learning were conducted, which makes me have a better understanding about deep learning.



(a) AUC - width (b) AP - width

Figure 6: The effect of different width

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 *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[2] J. F. Gemmeke, D. P. W. Ellis, D. Freedman, A. Jansen, W. Lawrence, R. C. Moore, M. Plakal, and M. Ritter. Audio set: An ontology and human-labeled dataset for audio events. In *Proc. IEEE ICASSP 2017*, New Orleans, LA, 2017.

[3] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.