Singing Transcription

Songrong Lee
National Taiwan University
r08922040@ntu.edu.tw

Pinyuan Chen
National Taiwan University
r08944024@ntu.edu.tw

Junyou Wang
National Taiwan University
b06902046@ntu.edu.tw

Yuhuei Tseng
National Taiwan University
r08922194@ntu.edu.tw

Abstract

In this project, we use four different models, ResNet-50, EfficientNet-B0, AlexNet, and RNN with pre-trained features, to convert a singing voice audio file into a list of MIDI notes, i.e. [onset, offset , pitch]. We compare the results and find out that RNN with pre-trained features performs the best on the task, which gives a comparable performance against other proposed methods in recent years. For the source code, please refer to our GitHub repository https://github.com/SongRongLee/adl-singing-transcription.

CCS Concepts: • **Computing methodologies** → **Machine learning**; • **Information systems** → **Information retrieval**.

Keywords: machine learning, singing transcription, multimedia information retrieval

1 Introduction

Singing transcription is the task of converting raw singing voice to sheet music. Currently, it is still a challenging task because of the diversity of singing voices. In our final project, we want to convert a singing voice audio file into a list of "notes", each note has three attributes: onset, offset and pitch. The pitch values are integers of semitones (MIDI numbers, for example, C2 corresponds to MIDI number 36).

2 Dataset

ISMIR2014 dataset [1] and MIR-ST500 dataset are used in this project. The former contains 38 music, while the latter contains 500 pop music files and not published yet. We use 450 songs from MIR-ST500 as the training set, and use the remaining 50 songs as the validation set. The testing set is the whole ISMIR2014 dataset.

2.1 Data Pre-processing

2.1.1 Spectrogram Concatenation

We resample the input audio file to 16000Hz and then compute three different magnitude spectrograms with a step size of 32 ms (512 sample points) and window sizes of 32 ms, 64 ms, and 128 ms (512, 1024, 2048 sample points) respectively. These magnitude spectrograms are then concatenated together in the frequency.

2.1.2 Frames Windowing

We don't feed the whole processed tensor of a single audio file into the model directly because we need frame-wise classification. Instead, for each frame, we concatenate six of its adjacent frames (3 before and 3 after the target frame) into a new tensor representing the feature of the target frame. The resulting tensor is of size [7, 1795]. We can think of a feature tensor of a frame as if it is an image of image classification problems.

2.1.3 Frames Labeling

In terms of the ground truth, we label each frame using three values: [onset_prob, offset_prob, pitch_class], where onset_prob is the probability whether a frame is an onset, offset_prob is the probability whether a frame is an offset, and pitch_class is an integer mapping midi number C2 to 0 and C6 to 48. A pitch_class equals 49 indicates no pitch at the frame. For example, an onset-only frame with pitch class #C2 will be labeled as: [1.0, 0.0, 1].

2.1.4 Data Instance Sampling

During the training phase, in each epoch, for each song, we randomly select one frame in that song as a training data instance. Each song contains about 8000 frames on average. Thus, theoretically, 8000 epochs of training implies a traverse of all frames in all songs.

The above mentioned pre-processing steps are for the cases in training CNN models. On the other hand, during the training phase of RNN, in each epoch, for each song, we randomly select 150 continuous frames in that song as a training data instance. During the testing phrase, we split the songs into chunks of 150 frames. Note that not every song can be divided exactly into chunks of 150 frames. We have two different ways to generate data depending on the way we do the post-processing. The first and trivial one is to discard the remaining part of the song where there are less than 150 frames. The second one is to append the last chunk with the zeros.

2.2 Post-processing

The output of AlexNet, ResNet-50 and EfficientNet-B0 is a vector of size 52: **[onset_logits, offset_logits, C2_logits, #C2_logits, . . . , C6_logits, no_pitch_logits]**. As for RNN,

its output is a matrix with size (chunk_size=150, 52): **[[onset_logits, offset_logits, C2_logits, #C2_logits, . . . , C6_logits, no_pitch_logits] ...]**

We process this result by applying sigmoid on onset_logits and offset_logits and applying argmax on C2_logits C6_logits to get the predicted pitch class. This is the result of a single frame of a song. Therefore, we have to collect the result of every frame in a given song. We then set up thresholds for onset and offset probability in order to determine if a frame is an onset or an offset, or neither. We generate our desired output result [onset_time, offset_time, pitch] by iterating through these frames, finding onsets and offsets based on the thresholds, and choose the most common pitch inside a note as the predicted pitch class.

Note that the thresholds are very crucial for our model to correctly classify onsets and offsets, resulting in a huge difference in the evaluation score. This is because of the imbalance of our input data, which has most frames not onsets or offsets. These thresholds should be tuned after the training is done due to the randomness of the input.

3 Approach

We have changed the network a little bit to fit our use case. The input channel number of all models is changed to 1. The output linear layer of ResNet-50, EfficientNet-B0 and AleNet will have an output of size 52. As for RNN, we add an extra output linear layer with an output dimension of size 52 after a GRU layer to fit our training purpose.

We use Adam as the optimizer and set the learning rate of 0.001. The output of a given audio file should be an array of notes (onset, offset, pitch). For onset and offset, we use BCEWithLogitsLoss, and for pitch class, we use CrossEntropyLoss. The loss is the sum of the three mentioned above.

3.1 ResNet-50

We use PyTorch's official implementation of ResNet-50 without pre-trained weights. It is based on He, Kaiming's paper[2] published in 2016. ResNet-50 is just a convolution network with convolution layers, pooling layers, and a fully connected layer. The key feature for this network to standout is the residual mechanism. The inputs of some layers get shortcuts to the later layers (some may include additional convolution layers. This mechanism is shown to be very effective in fixing degradation issues of large neural networks, and thus improves the performance of deeper neural networks while making them easier to be trained at the same time. In terms of training time, it requires around five hours to train 8000 epochs on a single NVIDIA TITAN RTX. The model file is of the size of 91MB.

3.2 EfficientNet-B0

EfficientNet-B0 is a CNN network that also uses the residual mechanism. It was first proposed in[4]. In this paper, the neural architecture search technique is used to develop this network. The main advantage of this network is its efficiency. Without large computation costs, it can still achieve state-of-the-art accuracy. Also, if more computational resources are provided, the paper shows that we can scale up the width, depth, and resolution of EfficientNet-B0 "with a set of fixed scaling coefficients" easily, and generate a larger model architecture such as EfficientNet-B7, which can achieve better accuracy without losing efficiency. However, with the limitation of computational resources, we can only use EfficientNet-B0 in our experiment. In the experiment, we use the implementation of EfficientNet-B0 without pre-trained weights from the GitHub repo "rwightman/gen-efficientnet-pytorch". In terms of training time, it requires around five hours to train 8000 epochs on a single NVIDIA GeForce GTX 1080 Ti. The size of the model file is only 16MB.

3.3 AlexNet

We used PyTorch's official implementation of AlexNet without pre-trained weights. It is based on Alex Krizhevsky's paper[5] published in 2012.The paper not only proposes that the depth of the model can improve performance but also uses the GPU to reduce computing costs. AlexNet was a breakthrough in deep learning in 2012. It has a total of eight layers. The first to fifth layers are Convolutional Layers for convolution operation and pooling; the sixth to eighth layers are Fully Connected Layers. The reason why it is so successful is to use ReLUs as activation functions (It was the first to implement Rectified Linear Units) and utilize dropout and data augmentation to avoid overfitting. In terms of training time, it requires around twelve hours to train 8000 epochs on a single NVIDIA TITAN X. The model file is of the size of 218.2MB.

3.4 RNN with Pre-trained Features

We use the pre-trained ResNet (3.1) features, which are extracted at the final two layers, and feed them into the 1 layer bidirectional GRU model. To go into detail, we use the pre-trained model to extract features for each frame of the chunk and concatenate them to get features that represent the chunk. After that, we would get a chunk feature with the shape [chunk_size, feature_size].

GRU was introduced in 2014 by Kyunghyun Cho et al[6]. GRU has 2 gates, reset gate and update gate respectively. The reset gate would decide how much the previous one hidden state could impact the new memory. The update gate would learn how much the previous one hidden state would pass to the current hidden state. We adopt GRU since its convergence rate and execution speed are faster than LSTM. Also, by the bidirectional mechanism, the model could preserve information from both past and future information compared to a unidirectional model.

In terms of training time, ResNet feature only requires around 17 hours to train 500 epochs on a 2 NVIDIA 1080 TI.

The model file is of the size of 6.5MB. EffNet feature only requires around 19 hours to train 500 epochs on a 2 NVIDIA 1080 TI. The model file is of the size of 4.2 MB. And, ResNet feature & EffNet feature requires around 35 hours to train 500 epochs on a 2 NVIDIA 1080 TI. The model file is of the size of 11MB.

4 Experiments

4.1 Evaluation

The evaluation measures we adopt here [1] are used to determine the performance of a singing transcriber, including "Correct Onset", "Correct Pitch" and "Correct Offset". Based on these measures, we can compute three metrics, "Correct onset (COn)", "Correct onset and pitch (COnP)" and "Correct onset, pitch and offset (COnPOff)". These metrics are widely used to evaluate the performance of an automatic transcriber. The tolerance of onset is set to 50ms, while the tolerance of offset is set to the larger one of 50ms and the 20% of the duration of the ground-truth note. The tolerance of pitch is set to 0.5 semitones. ‘

4.2 ResNet-50

According to Table 1, "COn F1-score" is a little bit better than the similar approach proposed by Fu Z.-S. and L. Su[3]. The main differences of their approach and ours are that we use a larger architecture of ResNet50 while they use ResNet18, and we apply an end-to-end model to extract onset, offset and pitch, while they only use ResNet to detect onsets and offsets. In terms of COnP and COnPOff, they are not standing out from other best methods in singing transcription, but are still comparable.

Table 1. Evaluation Results of RestNet-50

	Percision	Recall	F1-score
COnPOff	0.4519	0.4965	0.4711
COnP	0.6553	0.7087	0.6774
COn	0.7797	0.8362	0.8024

4.3 EfficientNet-B0

According to Table 2, the result is not as good as ResNet-50. However, considering the parameter number of EfficientNet-B0 (only about 4M, while the parameter number of ResNet-50 is 23M), the result is still not bad.

4.4 AlexNet

The shape of the original input tensor, [7, 1795], will disappear after many times of convolution operation and pooling. So, we use three different methods to resize it to [70, 1975]. The following are the three different results based on the same network.

Table 2. Evaluation Results of EfficientNet-B0

	Percision	Recall	F1-score
COnPOff	0.4431	0.4226	0.4303
COnP	0.6410	0.6020	0.6172
COn	0.8293	0.7645	0.7898

- Repeat 10 times in sequence
  We use PyTorch build-in method torch.repeat(), which will repeats the tensor along specified dimensions.
- Nearest neighbor Interpolation
  The nearest neighbor algorithm selects the value of the nearest point and does not consider the values of neighboring points at all, yielding a piecewise-constant interpolant.
- Bilinear Interpolation
  Compared to the nearest neighbor interpolation, bilinear interpolation refers to the four points around and determines how much to fill in based on the distance to each point as a weight.

Table 3. Evaluation Results of AlexNet

	Method	Percision	Recall	F1-score
COnPOff	Repeat	0.2095	0.2358	0.2183
	Nearest	0.3273	0.3611	0.3372
	Bilinear	0.3093	0.3203	0.3113
COnP	Repeat	0.5019	0.5358	0.5065
	Nearest	0.5784	0.6026	0.5742
	Bilinear	0.5894	0.5916	0.5820
COn	Repeat	0.6858	0.6976	0.6731
	Nearest	0.7524	0.7497	0.7265
	Bilinear	0.7560	0.7471	0.7398

Table 3 shows that Bilinear Interpolation has the best performance and repeat 10 times in sequence has the worst performance. Convolution operation and pooling will extract the information between pixels. If we just repeat data 10 times, like we copy the image 10 times, the features in the original image may disappear after many convolution operations and pooling. Besides, the reason Bilinear Interpolation has better performance than Nearest neighbor Interpolation is that the former makes data smoother. So after many operations, it will keep most of the features.

4.5 RNN with Pre-trained Features

Inspired by feature-level concatenation of AI CUP 2020 Mango Image Recognition Challenge baseline, we conduct 3 experiments, which are fixed model structure but only change the pre-trained CNN features, ResNet feature only, EfficientNet feature only and ResNet feature concatenated with EfficientNet feature respectively.

GRU shows great performance in grabbing context information. By using pre-trained CNN features in several different experiment settings (Table 4 and 5), we can see that models learn how to choose the useful feature during the training stage. In using ResNet's feature only and using Effnet's feature only experiments, the results show GRU uses the context information to perform classification better compared to the ResNet and EfficientNet models. Also, in the feature-level concatenation experiments, we get even better results than using a single feature in the RNN model. Comparing the data post-processing methods, we can notice that how to process would also have a noticeable impact on results.

Table 4. F1-score results on post-processing, discarding rest frames

Pre-trained model feature	COnPOff	COnP	COn
ResNet	0.4748	0.6632	0.7929
EfficientNet	0.4413	0.6376	0.7754
ResNet + EfficientNet	0.4819	0.6618	0.7835

Table 5. F1-score results on post-processing, predicting all frames

Pre-trained model feature	COnPOff	COnP	COn
ResNet	0.4816	0.6552	**0.8251**
EfficientNet	0.4813	0.6724	0.8200
ResNet + EfficientNet	**0.5149**	**0.6881**	0.8207

4.6 Comparison

Table 6 shows that RNN with pre-trained features achieves the best performance on every measure. It combines features extracted by ResNet-50 and EfficientNet-B0, which are two CNN models that also perform fairly well on this task, and makes predictions based on temporal information.

Apart from the RNN model, ResNet-50 performs best among three CNN models in terms of the evaluation score. This shows that with fewer parameters and thus smaller model size, ResNet and EfficientNet outperform AlexNet by using the residual mechanism and many other new techniques. Speaking of EfficientNet, which is a state-of-the-art image classification model, gives a comparable result to our best model ResNet50, by using nearly one-sixth of its model size. It does have the potential for better performance if we could train a larger architecture to compete in this task.

Although ResNet-50 is not a state-of-the-art image classification model right now in 2020, it still performs quite well on singing transcription tasks. Further experiments using larger models such as ResNet-152 or EfficientNet-B7 to perform singing transcription can also be conducted in the future if more computing resources and larger datasets are provided, which might further increase the performance potential.

Table 6. F1-score of Different Models

Model	size (MB)	COnPOff	COnP	COn
ResNet-50	90.1	0.4711	0.6774	0.8024
EfficientNet-B0	15.5	0.4304	0.6172	0.7898
AlexNet	218.2	0.3113	0.5820	0.7398
RNN	11	**0.5149**	**0.6881**	**0.8207**

Table 7. Comparison of Singing Transcription Results

Model	COnPOff	COnP	COn
Ryynänen [7]	0.3000	0.4700	0.6400
Tony [8]	0.5000	0.6800	0.7300
Fu & Su [3]	**0.5940**	NA	0.7860
Proposed	0.5149	**0.6881**	**0.8207**

5 Work Distribution

SongRong, Lee
ResNet-50, report organization and code structure

PinYuan, Chen
RNN with Pre-trained Features and report refinement

JunYou, Wang
EfficientNet-B0, data pre-processing and paper survey

YuHuei, Tseng
AlexNet and report refinement

6 Conclusion

In this final project, we apply several CNN and RNN models on the task of singing transcription. The RNN model with pre-trained features performs best, achieving 82.07% on COn measure, which is slightly higher than [3]. Such a result shows that the RNN model with CNN pre-trained model can deal with this task well. A CNN model can extract features from spectrogram, while an RNN model can make use of temporal information. By combining the advantage of these methods, the model can obtain sheet music from music files without making many mistakes. With such a model, it will be much easier for music lovers to get sheet music automatically.

References

[1] E. Molina, A. M. Barbancho-Perez, L. J. Tardón, I. Barbancho-Perez: "Evaluation framework for automatic singing transcription," in Proc. of the 15th International Society for Music Information Retrieval Conference (ISMIR 2014), pp.567-572, October 2014.

[2] He, Kaiming, et al. "Deep residual learning for image recognition." in Proc. of the IEEE conference on computer vision and pattern recognition, 2016
[3] Fu Z.-S. and L. Su: “Hierarchical Classification Networks for Singing Voice Segmentation and Transcription,” in Proc. of the 20th International Society for Music Information Retrieval Conference (ISMIR 2019), pp. 900-907, 2019.
[4] Mingxing Tan, Quoc V. Le: “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks,” arXiv:1905.11946, 2019.
[5] Alex Krizhevsky, et al. “imagenet classification with deep convolutional neural networks 2012.
[6] Cho, Kyunghyun, et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation." arXiv preprint arXiv:1406.1078 (2014).
[7] M. P. Ryynänen and A. P. Klapuri. Automatic transcription of melody, bass line, and chords in polyphonic music. Computer Music Journal, 32(3):72–86, 2008.
[8] M. Mauch, C. Cannam, R. Bittner, G. Fazekas, J. Salamon, J. Dai, J. Bello, and S. Dixon. Computer-aided melody note transcription using the tony software: Accuracy and efficiency. In Proc. SMC, 2015.