Generating Audio for Muted Piano-Playing Video using Deep Learning

XIA Junzhe
20493411
jxiaaf@connect.ust.hk

YANG Baichen
20493198
byangak@connect.ust.hk

HUANG Zeyu
20493631
zhuangbi@connect.ust.hk

Abstract

It is noticed that human playing the piano effectively generates a series of well-patterned actions, i.e. the position of hands and the keys and the depth of keys being pressed down. The notes that the piano generates strongly obey to this visual pattern. Hence, it becomes reasonable to recognize the visual patterns from piano-playing videos and reproduce the instrumental sounds using machine learning. In this project, we are going to propose a deep learning approach to effectively recognize key-press gestures and produce corresponding audios afterwards.

1. Introduction

The production and consumption of digital music and video are thriving in recent years, and related researches in music and video recognition have become rather important in related fields. Among these fields, one important problem is how to construct mapping from videos to songs, which is possible because the visual representation is a vital aspect of musics. To be more specific, how to conduct the conversion from video information to symbolic notation such as music scores or Musical Instrument Digital Interface (MIDI) file. This problem is quite realistic for music lovers as it is hard to replay a music video without music transcription support.

Our target is to generate music transcription and corresponding audio from piano-playing videos without any sound information. Previous works have been done towards this problem by implementing traditional computer vision methods for this task. However, evaluation results show that these methods are either not accurate or easy to be influenced by environmental conditions. So in this report, we propose a more robust solution for piano video recognition and music re-generation.

To accomplish this task, we divide it into three stages.

1) Keyboard Extraction As the keyboard's location differs in different videos, the first step is to develop a method extracting the whole keyboard from the background and conduct rectification on it. The extracted keyboard will be fed into the second step.

2) Keypress Recognition After the keyboard area is extracted, our core task is to recognize finger-key correspondence and determine whether the keys are pressed or not. To finish this recognition, we need to focus on the piano keys' arrangement pattern as well as the illumination circumstance. However, since there are often reflections, shadows and other visual noises cast on the keyboard, it is rather difficult to develop an algorithm based on traditional computer vision method. Therefore, a robust algorithm is required.

3) Velocity Evaluation The velocity information is essential in audio emotion expression. With a high velocity value, the song conveys stronger feelings, and vice versa. Since the velocity is a kind of dynamic information, our algorithm need to analyze several neighbor frames simutaneously in order to get the estimation.

We have developed methods to solve these three parts and received a quite good result. For the *Task 1*, we've tried both traditional computer vision methods and deep learning methods, finally determined to use the *SIFT* method [6] to achieve a better outcome. About the *Task 2*, we trained two different neural network models for black and white keypress detection. Both neural network models adopt VGG-like structure[7] with two or three layers structure as the key size is rather small for deeper convolutional neural network to learn. And lastly for the *Task 3*, we use the LSTM structure [5] and feed in neighbor frames of a video to perceive the change of the keypress as well as the velocity.

We have achieved quite acceptable results. In keypress recognition part, our models gained $\sim$ 98% accuracy on test set, which is higher than the currently best CNN-SVM model [1]. Also, in the velocity evaluation part, we achieved an average L1 Loss of $\sim$ 12 (black key) and $\sim$ 17 (white key) on test set.

2. Related Work

Previous works have been done in related area and some of them are targeting at this specific problem. In this section we will give brief introduction for them and discuss their strengths and drawbacks.

One previous solution was proposed in Gorodnichy and

Yogeswaran's paper [3] where traditional computer vision method is utilized to complete the keyboard and hand extraction as well as the gesture recognition tasks. Detaily, this solution divides the problem into three parts: Keyboard image detection, Hand detection and Finger detection. It compute the keyboard boundary according to the assumption that white keys are surrounded by nonwhite areas, i.e. the division algorithm is based on the contrast information. And for the hand and finger detection, it utilized the background subtraction method and crevice-detection method. This solution achieved this task only with traditional computer vision methods, whose cost is lower than neural network model as the latter one needs large dataset and training. However, these methods required a frame where there are no hands on the keyboard because the flood fill algorithm will fail if there is a hand involving in. Also, since illumination conditions may differ, this traditional computer vision method doesn't perform quite well under different circumstances.

Another solution towards this problem was proposed by Akbari et al. [1] They proposed a system of automatically annotating piano-playing video using convolutional neural networks (CNNs) and two separate binary support vector machines (SVMs). Though a high accuracy has been achieved in keyboard and hand detection, this solution tackles issues according to some artificially formulated rules related to the visual nature of the piano keyboard, which is hard to prove its correctness in most cases. Also, since it applies CNN method in all stages of the problem, the computational cost is rather high, which is unavoidable.

3. Dataset

In this section, we will talk about our dataset source and pre-processing.

We adopt our dataset mainly from two sources: **modified from the dataset that was been used in previous work done by Akbari [1]; our own record**. Both datasets are formed by two parts: muted piano-playing videos and corresponding MIDI file, where contains all the pitch and velocity information.

3.1. Dataset Source

3.1.1 Dataset from Previous Work

First part of our dataset is obtained from a previous research [1]. It consists of piano playing videos captured in different situations which may lead to enhancement of model's strength:

Camera Position As variations in position may lead to keyboard detection failure, this dataset's recording position contains three posibilities: $+45,\ 0,\ -45$ degrees from vertial.

Pianists This dataset includes 14 different players with different hand sizes, playing styles and skin colors, which will increase robustness of the model.

Positive and Negative Examples This dataset also include both positive and negative examples, where negative stands for moving the hands over the keyboard without pressing any key.

# of Frames	# of Black	# of White
70540	~600000	~400000

Table 1: Stats of Previous Dataset

3.1.2 Our New Dataset

However, since the first part of our dataset doesn't include velocity information, we recorded our own dataset [1] additionally to train the network. Despite of velocity information, our dataset also has the following several features:

High Resolution Our dataset is recorded by $1080P$ camera. The keyboard resolution increased by 70% than the previous dataset. We downsize the keyboard images into 884×106 before feeding into the model, which is still even higher than the resolution of the first part of our dataset without downsizing.

Music The first dataset only contains sequentially key-press sound track, which cannot represent real music piece. In our dataset, we played and recorded several master pieces to simulate real piano-playing videos.

# of Frames	# of Black	# of White
22266	14536	27843

Table 2: Stats of Our Dataset

3.2. Dataset Preprocessing

3.2.1 Conversion from/to MIDI format

To get a trainable dataset, we firstly align the beginnings and the ends of videos and MIDI audios and sample frames out of both media files in a sampling rate of 25 fps. Then, we generates a $N \times 88$ label for each video file recording the status of each piano key in each frame, where N stands for the number of frames in that video.

The output of our method is a matrix of similar shape. The matrix undergoes a reverse process to transform into a MIDI file with the same sampling rate.

3.2.2 Single Key Separation

After we get the standardized keyboard image 4.1, we need to further separate it into single white keys and single black

[1] This dataset is available to the public. Due to the large file size, if you are interested, please contact us directly via email.

keys for training.

As for white keys, we equally divide the width of the image into 52 parts in the sense that all white keys have the same width.

$$White_i = Image[0 : Height, i\times\frac{Width}{52} : (i{+}1)\times\frac{Width}{52}]$$

As for black keys, since different keyboard has different black keys layout. We extract the top 5 rows of the standardized keyboard image and use Otsu's Thresholding Algorithm to convert it into a binary image. Since most of cases, players' hands will not overlap with the very top part of the keyboard, we can expect that the image extracted has a black-white alternating pattern:

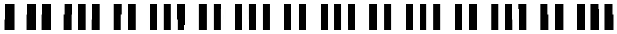


Figure 1: Expected Extracted Image

Then we take the average value of pixels in each columns, then every white-to-black pixel has exactly the x-coordinate of a black key's left boundary, every black-to-white pixel has the x-coordinate of a black key's right boundary.

$$Left_i = \{x : avg_x > 0 \text{ and } avg_{x+1} = 0\}_i$$
$$Right_i = \{x : avg_x = 0 \text{ and } avg_{x+1} > 0\}_i$$

After we get the left and right boundary of each black key, we can localize every black key as following:

$$Black_i = Image[0 : Height, Left_i : Right_i]$$

4. Methods

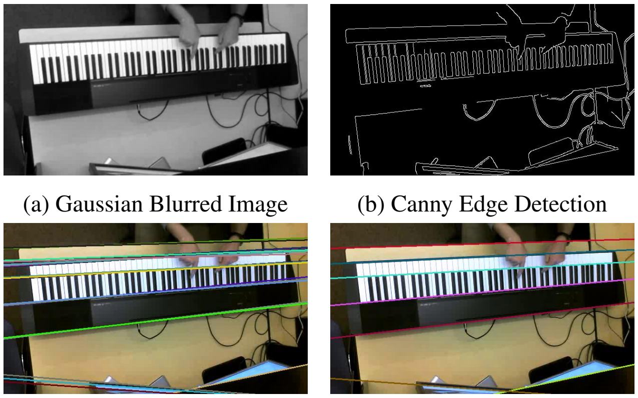
In this section we will talk about methods tried and their pros and cons. As discussed in Section 1, we divide this problem into three stages, keyboard detection, keypress recognition and velocity evaluation.

4.1. Keyboard Detection

Towards the keyboard detection problem, we've tried three methods.

4.1.1 Hough Line Detection

Firstly, we adopted the traditional CV method Hough Line Detection into this task, which is shown in Figure 2. Firstly, we can obtain the edge information of the picture by conducting Gaussian Blurring and Canny Edge Detection on it. Next, we apply Hough Line Transformation [2] on the preprocessed picture to detect boundary lines candidates. And since there'll be several repeated boundary line detected, we prune those repeated lines according to their slopes and positions. Finally, we determine the exact two parallel lines that bound the keyboard by examining the black and white pixel location pattern.



(a) Gaussian Blurred Image

(b) Canny Edge Detection

(c) Hough Line Transformation

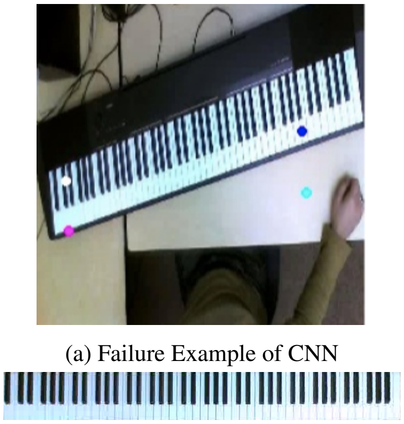
(d) Prune Repeated Lines

Figure 2: Hough Line Detection Method

However, this method has some major drawbacks. For one, the computational complexity of Hough Line Transformation is $O(n^3)$, where n is the number of pixels. So the computation is quite expensive. In addition, to complete such detection, a frame without hand on the keyboard is needed, which is unobtainable for some videos. Therefore, we resort to the second choice, using Deep Convolutional Neural Network.

4.1.2 CNN Detection

We next tried Convolutional Neural Network to detect the keyboard location. As the dataset doesn't offer us information about the exact location of keyboard, we manually labelled coordinates of the keyboard and feed them into a pre-trained ResNet-18 [4] model for coordinate regression.



(a) Failure Example of CNN

(b) SIFT Result

Figure 3: Comparison between CNN and SIFT

However, since the dataset is highly homogeneous because they are extracted from 71 videos, the model tends to build a mapping between image features to 71 fixed coordinate sets, which made the models performance on test set quite undesirable. A failure example is shown in Figure 3a

We therefore turn to our third choice, the SIFT method.

4.1.3 SIFT Detection

Our finalized solution on this task is Scale Invariant Feature Transform (SIFT) [6].

For each image in the dataset, we extract SIFT descriptors $V_i^{1\times 128}$ $(i = 0, 1, 2, 3)$ for pixel patches containing the four keyboard corners in each image using the coordinates we've labelled in the previous method.

In test time, we derive the four keyboard corners by finding the point whose SIFT descriptors are most similar to the ground truths, i.e.

$$P_i^* = \arg\min_P \sum_{k=0}^{64} \sqrt{\sum_{j=0}^{128} (V_j^P - V_{ij}^k)^2}$$

The result shows that this method is quite robust and accurate in determining the keyboard location. Under most circumstances, the method can extract the keyboard correctly and entirely. A comparison example is shown in Figure 3

4.2. Keypress Recognition

Now that we have obtained the extracted keyboard image from background, we are able to further conduct keypress recognition task on it. We will develop CNN models to accomplish this task.

4.2.1 Loss Function

The loss function we used is **Binary Cross Entropy Loss**, which can be described as following:

$$\begin{aligned} w^* = &\arg\min_w - \sum [y_i \log w(x_i, \theta) \\ &+ (1 - y_i) \log(1 - \log w(x_i, \theta))] + \lambda ||w||_2 \end{aligned}$$

The reason is that we are targeting at a binary labelling problem. That is, our dataset $D = \{(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\}$ where $x_n \in \mathbb{R}^{C\times H\times W}$ is the images and $y_n \in \{0, 1\}$ is the predicted probability of begin pressed, and data pairs are independent of each other. After defining our loss function, our next step is to process the data and set up the network model.

4.2.2 Whole Keyboard Detection

Our first attempt is to put the whole keyboard and corresponding frame note label into the neural network, trying to recognize both the pressing pattern as well as key location by one network. We utilized a deep convolutional neural network solution for this task since it could preserve spatial information of the picture and focus its attention on some particular region. The set our CNN structure is VGG-like [7], and the design is shown in the Figure 4a.

This method doesn't require data processing step 3.2.2. However, we failed to obtain a good result with this approach. One major problem may be that in our training set, some keys are frequently pressed while some are rarely pressed. This inbalance of frequency could provide the model with wrong attention. Also, as hands are small compared to the whole keyboard, the network tends to focus on other factors such as illumination conditions instead of hand motions, which may lower the accuracy.

4.2.3 Single Key Splitting

To help direct the model's attention, in our next attempt we split the input image into 88 segments using the method elaborated in 3.2.2, Since we have resized and skewed the images into standard 884×106 rectangles, the splitting process yields a rather promising result. Finally, we add some "tolerance" by widening the cropped area by 2-4 pixels on its left and right sides to avoid the cropping area being to tight.

We have trained two models for black and white keys respectively, each of the same structure shown in 4b.

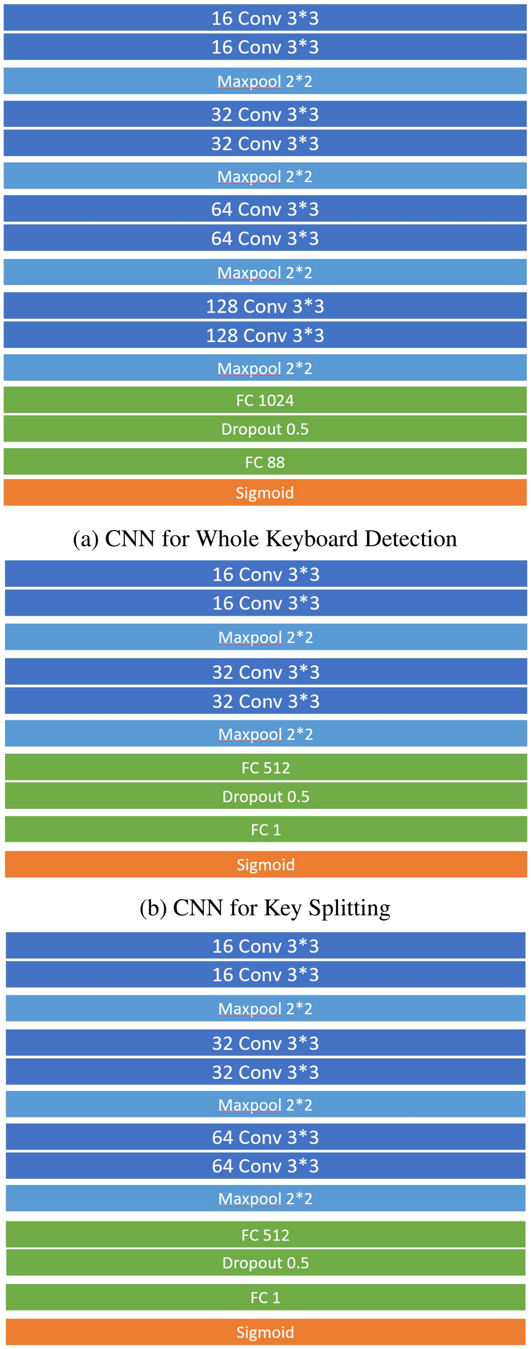
This model works quite well on our dataset, and achieves higher accuracy than the previous method as we manually help direct the model's attention and eliminate the aforementioned frequency inbalance issue.

4.2.4 Key Bundling

Lastly, we also introduce a second splitting stretegy called "bundling", where in this mode the "tolerance" is enlarged to include some neighbor keys. In other words, we will produce relatively wide images of several keys when the centered key is our training target.

In our implementation, we make the number of neighbor keys to be bundled a variable hyperparameter. Since bundling too many keys tends to incur more noise, after some experiments we determined the bundling number to be $B = 3$, including exact one key on both sides. (See Figure 5)

The model we use to train bundled key images are shown in Figure 4c. In our experiments, this approach outperforms the single key splitting approach most of the time by ~ 0.1. Detailed comparisons are listed in 5.1.5.



(a) CNN for Whole Keyboard Detection

(b) CNN for Key Splitting

(c) CNN for Key Bundling

Figure 4: CNN structures for Keypress Recognition

(a) Single Key Split

(b) Key Bundling

Figure 5: Comparison between two splitting modes

4.3. Velocity Evaluation

Last part of our task is to evaluate the velocity of each keypress. In MIDI file, the velocity is recorded as an integer value ranges from 0 to 127. So our objective is to detect motion in "pressed" frames and neighbor frames and make a prediction on the velocity value of that key press.

Since this prediction involves extracting features in a sequence, we use LSTM to tackle this problem.

4.3.1 LSTM Architecture

In the sense of that velocity is related to how fast the keypress depth changes, which is also an important feature to determine whether a key is pressed, we used a CNN model pretrained in keypress detection 4.2 as the feature extractor for each input image. Then a sequence of feature tensors will be fed into a two-layer LSTM network with hidden size of 256, generating a single logit value.

$$Velocity = Sigmoid(logit) * 127$$

After gating and rescaling this value, we will derive the final predicted velocity ranging from $[0, 128)$.

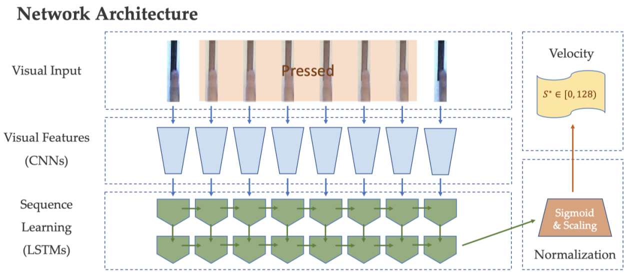


Figure 6: LSTM Architecture for Velocity Evaluation

4.3.2 Loss Function

The loss function we used is **Mean-Squared Loss** which can be described as following:

$$w^* = \arg\min_w [y - w(x, \theta)]^2 + \lambda ||w||_2$$

While training, we used sigmoid $(logit)$ as the final output of the network instead of sigmoid $(logit) \times 127$.

5. Experiments

In this section we will talk about our experiment settings including the hyperparameters, and our current results.

5.1. Keypress Recognition

The whole keyboard network is deprecated in an early stage of our experiments due to its shortcomings mentioned in 4.2.2. Hence, the following experiment details only concern the single key and bundled key approaches, and there are four models in total.

5.1.1 Hyperparameter Tuning

All four models are trained 200 epoches with Adam optimizer with learning rate $1e-3$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1e-8$, and a learning rate decay by a multiplier of $\gamma = 0.5$ every 10 epochs. We use a batch size of 32, and an L2 penalty of 0.1 on the model parameters.

5.1.2 Evaluation Metrics

To evaluate our keypress recognition performance, apart from the common accuracy, we also introduce two finer criterion: *Precision* and *Recall*.

$$Precision = \frac{TP}{TP + FP}$$
$$Recall = \frac{TP}{TP + FN}$$

where

TP = # pressed keys predicted as “pressed” (True Positive)

FP = # unpressed keys predicted as “pressed” (False Positive)

FN = # pressed keys predicted as “unpressed” (False Negative)

We expect both metrics to be high. However, an increase in one often indicates a decrease in the other. Thus, we introduce another commonly used metric named *F1-Score*:

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

to describe the recognition performance.

5.1.3 Misleading Training Result

In the very early stage, we use the raw dataset to feed the model during training and get following result:

Precision	Recall	Accuracy
1.39 %	2.04 %	98.5%

Table 3: Misleading Training Result

This result somehow revealed a very serious problem: most of the samples in the dataset are unpressed key images. So the model only need to output “unpressed”, then a low loss and a high accuracy can be achieved. However, such a model is useless at all. Therefore, we need to perform some preprocess on dataset to make the training process meaningful.

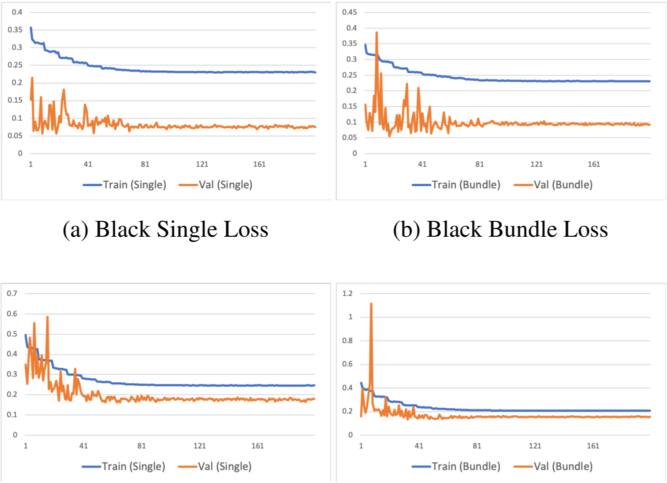
5.1.4 Reduce Dataset Unbalance

To solve this dataset unbalance problem, we randomly took out some of the negative samples because most unpressed key images are almost the same, which are solely empty keys with no hand on it. Also, we upsampled those positive samples by simply duplicating them.

After applying this trick, the dataset contains almost the same number of positive samples and negative samples. We conducted some experiments with this improved dataset and concluded that this preprocess step prevented the model from only outputing only one result.

5.1.5 Results

We have reached $90\%+$ training and validation accuracy on both black keys and white keys after applying the preprocessing mentioned above.



(a) Black Single Loss (b) Black Bundle Loss

(c) White Single Loss (d) White Bundle Loss

Figure 7: Stats during Training Process

Both Single and Bundle case splitting achieves quite good results. The validation set and test set results are listed below:

	Precision	Recall	Accuracy	F1 Score
Bundle-B	74.4 %	70.9 %	99.6%	72.6%
Bundle-W	75.6 %	72.7 %	98.3%	74.1%
Single-B	67.7 %	67.0 %	99.5%	67.3%
Single-W	74.1 %	72.8 %	98.3%	73.4%

Table 4: Validation Result

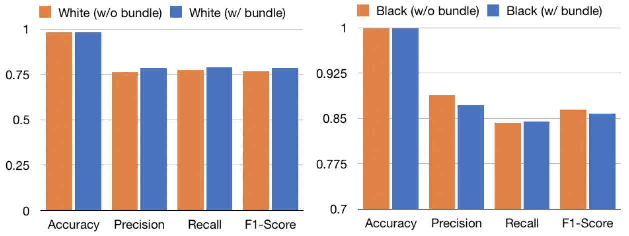


Figure 8: Comparison between Bundle and Single (val set)

	Precision	Recall	Accuracy	F1 Score
Bundle-B	87.2 %	84.5 %	99.9%	85.8%
Bundle-W	78.4 %	78.8 %	98.2%	78.6%
Single-B	88.8 %	84.2 %	99.9%	86.4%
Single-W	76.3 %	77.4 %	98.1%	76.8%

Table 5: Test Result

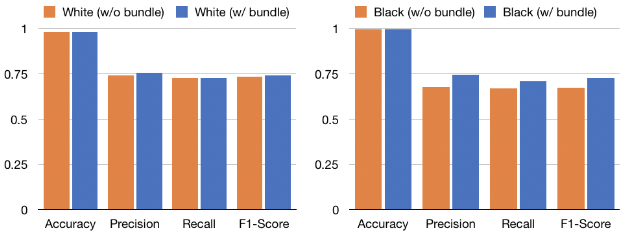


Figure 9: Comparison between Bundle and Single (test set)

Comparing to the previous work done by Akbari[1] whose accuracy have already been quite good, our method achieves even better results on keypress recognition.

	Our Result	CNN-SVM
Black-key	99.6%(Val) 99.9%(Test)	94.6%
White-key	98.3%(Val) 98.2%(Test)	95.5%

Table 6: Accuracy Comparison

5.2. Velocity Evaluation

As for the LSTM network for velocity evaluation, we adopted pretrained CNN models for single white keypress and single black keypress as feature extractors. While training, we froze CNN networks to mainly improve the performance of the LSTM model.

5.2.1 Hyperparameter Tuning

Both black and white models are trained 60 epoches with Adam optimizer with learning rate $5e-3, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1e-8$, and a multiplier decay of $\gamma = 0.05$ for every 15 epochs. We use a batch size of 1, and an $L2$ penalty of 0.1 on the model parameters.

5.2.2 Evaluation Metrics

To evaluate the velocity evaluation performance, we use average L1 Loss to measure the predicted velocity values and ground truths.

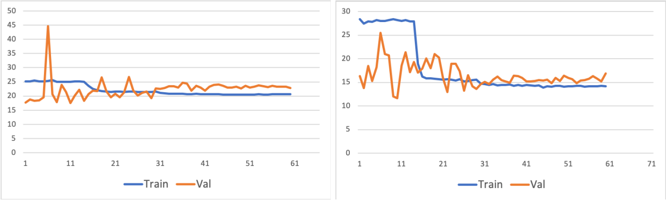
$$Average\ L1\ Loss = \frac{1}{N}\sum_{i=1}^{N}|\widehat{y_i} - y_i|$$

5.2.3 Results

In the model for white key velocity evaluation, we reached an average L1 Loss of 17.6093 on test set with the best model. In the model for black key, an average L1 Loss of 11.6546 is reached on test set with the best model.

Color	Val Set	Test Set
Black	11.6546	12.8342
White	17.6093	16.9471

Table 7: LSTM Training Result (L1 Loss)



(a) White L1 Loss History (b) Black L1 Loss History

Figure 10: Stats during Training Process

However, we believe there still exists large gap between our models and ideal models for velocity evaluation. Because considering the velocity only ranges from 0 to 127, a L1 Loss > 10 is far from a neglectable difference.

According to the loss history diagrams, we find that the curve did not go down after the first drop of learning rate. To tackle this problem, we tried some other methods. However, those methods have not brought us satisfactory results.

5.2.4 Failed Attempts

For the sake of getting lower loss in both train set and val set, we have tried to:

- Freeze LSTM and activate gradients on CNN after the first 60 epoches, then continue to train this model - This method failed as it finally brought the loss back to the leval after the first 60 epoches after some fluctuations.
- Use lower learning rate to continue training LSTM part - This method does decreased the loss on train set but significantly increased the loss on val set, namely, it led the model to an overfit one.

5.3. Demo Video

After all the models were finalized, we regenerated audio for a testing video that we recorded.

According to the video, compared to the original audio, we can notice that our models are able to recognize almost all pressed keys. However, some false positive recognitions happens when a key is going to be pressed or a finger is closely above a key. Also, some notes with long duration are recognized into several seperate notes by our models as this key are classified as unpressed wrongly in the pressed duration.

We have not added any make up on those flaws as they can reveal weaknesses of our models - hard to correctly distinguish some false positive pressings and unstable performance on long notes recognition. We believe these aspects are what we are going to work on in the future.

The video is currently available online.[2]

Figure 11: Preview on Demo Video

6. Conclusion and Future Work

In this paper, we've introduced a method to tackle the audio generation problem and achieves quite good performance. Through our method, we could get the music score from a muted piano-playing video with a high accuracy up to $\sim 98\%$. And by this music score as well as the velocity information predicted, we are able to re-generate the audio with emotion.

A demo video made by us is currently available online for reference. Comparing to previous solutions, our method achieves either better accuracy or becomes more robust under most of circumstances.

In the future, we plan to refine our velocity evaluation structure so that our generated audio can be more vivid and realistic. Also, we hope to implement a real-time recognition system to realize immediate pitch detection.

References

[1] M. Akbari, J. Liang, and H. Cheng. A real-time system for online learning-based visual transcription of piano music. *Multimedia Tools Appl.*, 77(19):25513–25535, Oct. 2018.

[2] R. O. Duda and P. E. Hart. Use of the hough transformation to detect lines and curves in pictures. Technical report, SRI INTERNATIONAL MENLO PARK CA ARTIFICIAL INTELLIGENCE CENTER, 1971.

[3] D. Gorodnichy and A. Yogeswaran. Detection and tracking of pianist hands and fingers. 2006.

[4] 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.

[5] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[6] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[7] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[2] https://drive.google.com/open?id=14rtqHUi0hxr7s5JEfRmYp-EL-IPaK8mZ