Combining CNN and Classical Algorithms for Music Genre Classification

Haojun Li (haojun) [1], Siqi Xue (sxue5) [2], Jialun Zhang (jzhang07) [2]
[1] Department of CS, Stanford University, [2] ICME, Stanford University

***Abstract*—In this study, we combine Convolutional Neural Networks and classical classification algorithms such as Softmax Logistic Regression, SVM, GDA, and Random Forests for music genre classification. By training a Dilated CNN as a feature extractor for classical algorithms, we obtain a comparison of different classical algorithms using different input features (which corresponds to the different layers of the Dilated CNN). We find that this method allows us to greatly improve the performance of classical algorithms, even allowing them to exceed the the performance of the Dilated CNN. We noticed that classical algorithms have a regularization effect so even with a non-optimal CNN we can improve its performance by feeding the layer outputs as features to classical algorithms.**

I. Introduction

Music is becoming more and more easy to access through internet and musical apps. With the increasing amount of music available digitally online, there is a growing demand for systematical organization of audio files and thus a rising interest in automatic music genre classification. Moreover, detecting and grouping music of similar genre is a keen part in music recommendation system and playlist generator.

There are indeed widely accepted rules in music theory that help human classification, such as chords and rhythmic structures, instrumental arrangement, etc. However, in general, musical content is complex and music genres are not well defined, making it a challenging machine learning problem, and distinguish it from many other classification problems.

The goal of this paper is to improve the performance of *classical algorithms*, including Logistic Regression (LR), Gaussian Discriminant Analysis (GDA), Random Forest (RF) and Support Vector Machine (SVM), by combining them with a *dilated convolutional neural network* (Dilated CNN, or DCNN). The methodology is structured as follows. We transfer the raw music inputs in wave format into 2D MFCC features at each timestamp. As a baseline, the 2D arrays are flattened and feature selected through PCA. This is then fed directly into the classical algorithms described above. To improve prediction, we pass the same 2D arrays into a 2-layered, 1-dimensional Dilated CNN, and train it for reasonable performance. The activation of each convolution layer are extracted as inputs to the classical classification algorithms. All outputs of the models are probabilities from a softmax layer denoting the probability of the example being in a specific genre. [1]

II. Related Works

Music genre classification, as a branch of audio and speech processing, has been studied by many. A standard approach consists of two steps, feature extraction and classification.

Most of the genre classification studies focuses on finding the best set of temporal features, transformations, and filters that best represent the music.[7] The author of the dataset we are using also attempted to find the set of features that best represent a music.[5] Other studies will try to find combinations of well known music theories such as rythm analysis to add new features to the classification problem.[6] We believe that this significantly limit the performance of models because these features are ultimately extracted by humans and we will be missing some important features that could be extracted by a neural network.

Other studies have tried to use some AI/Machine learning techniques such as Hidden Markov Model to classify music genres[9], and even SVM[3]. However, they still have limited performance. In recent years, deep learning and neural networks have also been widely applied to classifications problems, including music genre classification. More specifically, using CNN as a music feature extractor was studied by T. LH. Li, A. B. Chan, and A. HW. Chun[4]. They used MFCC audio representation and trained a music pattern extractor to classify music genre. There are also LSTM music genre classification works being done [8] but mostly focused on lyrics.

In this study, we will build on top of the works done before and see if we can improve them by combining classical algorithms with neural networks.

III. Dataset and Representations

The dataset we used is taken from GTZAN [2], which is the same dataset used by G. Tzanetakis and P. Cook in their paper[5]. It consists of 1000 30-second audio tracks,

[1] Disclosure: We will use the same technique to extract features for our CS230 project, but we did not use the same data set and we did not use the same input/output or code

which break down into 10 genres. Each genre contains 100 pieces of music in .wav format. We will use only 5 genres from the 10, namely *classical, hiphop, metal, pop*, and *blues*. Therefore we have only 500 music pieces. We then split them into 400 training pieces and 100 test pieces.

Using the `LibROSA` library in Python, the data is pre-processed into MFCC (mel-frequency cepstral coefficients) features. MFCC features are commonly used in sound processing and music classification. It allow us to represent each music wave file as a 2D `numpy` array (Figure III.1), with its x axis as time, and the y axis as MFCC features.

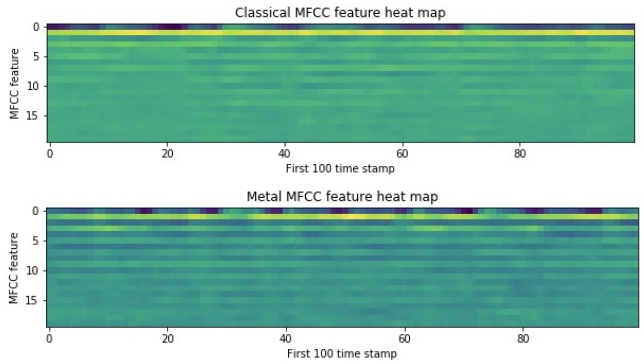


Figure III.1: 2D MFCC array of a classical and a metal music piece from the dataset, with the horizontal axis as time, and the vertical axis as MFCC values.

The figures above shows a heat map of classical and metal MFCC features. As we can see there are already some interesting but subtle differences that we can see with our eyes.

IV. Models

Before any training experiments, the dataset is split into train and test sets with a 80/20 ratio, and they will stay the same throughout the whole project.

A. Classical Baseline Models

We investigated the performance of four classical classification algorithms: Softmax Regression (SR), Gaussian Discriminant Analysis (GDA), Random Forest (RF) and Support Vector Machine (SVM). For the equations in this section, we will always use x to denote the input feature and y to denote the labels. Moreover, we will use $x^{(i)}$ to denote the training examples and $y^{(i)}$ to denote the true class for $x^{(i)}$, and $\hat{y}^{(i)}$ as the predicted class for $x^{(i)}$

- For **softmax logistic regression**, the model is

$$h_\theta(x) = \frac{1}{\sum_{j=1}^{K} \exp\left(\theta^{(j)\top} x\right)} \begin{bmatrix} \exp\left(\theta^{(1)\top} x\right) \\ \exp\left(\theta^{(2)T} x\right) \\ \vdots \\ \exp\left(\theta^{(K)\top} x\right) \end{bmatrix},$$

  with the corresponding cost function

$$J(\theta) = -\sum_i \sum_k \mathbb{1}\{y^{(i)} = k\} \log \hat{y}_i$$

  where $\mathbb{1}\{y^{(i)} = k\}$ is the indicator of whether a class actually belongs in class k and $\hat{y}_i = h_\theta(x)_i$ is the predicted probability of whether a class belongs in k.

- For **GDA**, the model assumes that the class labels y is a multinomial distribution with 5 values with parameters $p(y = k) = \phi_k$. Moreover, we assume that $x|y = k \sim N(\mu_k, \Sigma)$, where $N(\mu_k, \Sigma)$ is the multivariate Gaussian with the density function

$$\frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}\left(x-\mu_k\right)^T \Sigma^{-1}\left(x-\mu_k\right)\right).$$

  The parameters μ_k, ϕ_k and Σ are calculated by maximizing the log-likelihood of the given data

$$\begin{aligned} \ell\left(\vec{\phi}, \vec{\mu}, \Sigma\right) &= \log \prod_{i=1}^{m} p\left(x^{(i)}, y^{(i)}; \vec{\phi}, \vec{\mu}, , \Sigma\right) \\ &= \log \prod_{i=1}^{m} p\left(x^{(i)} | y^{(i)}; \vec{\mu}, \Sigma\right) p\left(y^{(i)}; \vec{\phi}\right). \end{aligned}$$

- A comprehensive review of **random forests** can be found in sections 9.2 and 15.1 in [10] . Roughly speaking, random forests is an ensemble method that applies bootstrap aggregating (bagging) to decision trees so that the final model is better at avoiding overfitting. The error that we choose for the splitting method is the Gini index. For a particular region R with N observations, it is defined as

$$\sum_{k=1}^{K} \hat{p_k}(1 - \hat{p_k}),$$

  where

$$\hat{p_k} = \frac{1}{N} \sum_{x_i \in R} I(y_i = k)$$

  is the percentage of examples in class k in the region R. In our implementation, we choose a maximum depth of 7. We observed that choosing a greater depth with allow us to fit to the training data with 100% accuracy, but this will also result in high test error.

- Traditionally, **SVM**s work naturally as two-class classifiers. In particular, we are solving the following optimization problem:

$$\begin{aligned} &\min_{w,b} \frac{1}{2}\|w\|^2 \\ &\text{s.t. } y^{(i)}\left(w^T x^{(i)} + b\right) \geq 1, \quad i = 1, \ldots, m \end{aligned}$$

  For the multiclass classification problem with $|C|$ classes, we extend the two classifier by constructing $|C|(|C|-1)/2$ classifiers for each pair of classes. Then for each $x^{(i)}$ we choose its class by setting $y^{(i)}$ to be the class that $x^{(i)}$ appeared in the most number of times. Finally, we note that in practice we use a kernalized version of SVM, where the kernel is given by the radial basis function

$$K\left(\mathbf{x}, \mathbf{x}'\right) = \exp\left(-\gamma \left\|\mathbf{x} - \mathbf{x}'\right\|^2\right),$$

  and γ is scaled inversely by the number of features and the standard deviation of the inputs. This will map

the input features to a high dimensional feature space, allowing us to generate more complex boundaries.

As a baseline, we flattened the raw 2D MFCC arrays into 1D vectors, and feed them to the 4 above-described classification algorithms. The flattened vectors are of length 25800. For classical algorithms, a 25800 dimensional input can be quite high and susceptible to overfitting. A preliminary idea is to apply principle component analysis (PCA) to extract the top 50 principle components. However, we must take caution when applying PCA since it does not always improve the test accuracy. Classification results for both the origin input and the input after PCA is compared in the results section.

B. Dilated Convolutional Neural Network (DCNN)

After we have established our baseline model, we then trained multiple dilated Convolution Neural Networks and compared results. The main difference between a Dilated CNN and original CNN is that each input unit is seen by a filter with gaps between them. A simple illustration for DCNN can be found in Figure IV.1. With a dilation rate of 4 and filter size 2, each filter will see 2 input units that are 4 distance apart.

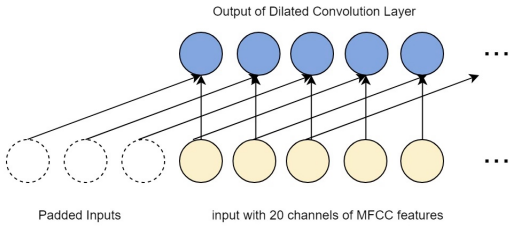


Figure IV.1: A dilated convolution layer, illustrated with filter size 2 and dilation rate 4. The light yellow nodes have 20 channels of MFCC features, and each blue node has number of channels equal to the number of filters used.

The architecture that we end up with is shown in Figure IV.2.

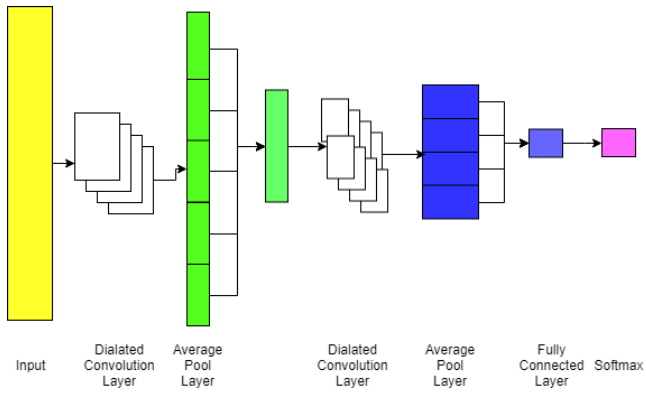


Figure IV.2: Dilated convolution neural network architecture used in this project.

Simply put, the architecture has a 1D dilated Convolution Layer followed by a average pool layer (with dropout) and a batch norm layer, forming a dilated convolution unit. This is then repeated with different parameters, and finally connected to a fully connected layer with softmax activation, which outputs a probability among 5 classes.

For optimization, we use the categorical cross entropy loss defined by

$$J(\Theta, B) = -\sum_i \sum_k \mathbb{1}\{y^{(i)} = k\} \log \hat{y}_i,$$

where Θ is the weights in our neural network and B is the bias terms. We optimize over Θ and B using the `adam` optimizer, which combines gradient descent with momentum and RMSprop.

We tried many models. The table of comparison is shown in Table IV.1.

Model	Training Accuracy	Test Accuracy
C M F O	0.8475	0.78
DC M F O	0.94	0.8
C A F O	0.94	0.81
DC A DC A O	0.99	0.86

Table IV.1: Comparison of different model architectures: C denotes a convolution layer, DC denotes a dilated convolution layer, M denotes a max pooling layer, A denotes an average pooling layer, and F denotes a fully connected layer; O denotes the output of the model.

C. Combining DCNN and Classical Algorithms

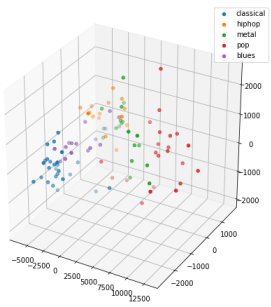
Lastly and most importantly , we will use the dilated CNN model as a feature extractor for the classical classification algorithms.

Having our DCNN model trained, we take the activation outputs of the two convolution layers, flatten them, and feed them into the four classical algorithms that we described above.

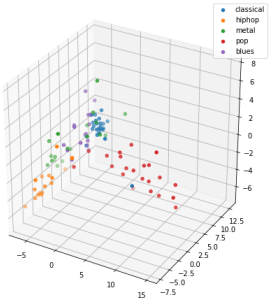
To summarize, we will use (1) flattened MFCC vector, (2) PCA reduced MFCC vector, (3) convolution layer 1 output (light green in figure IV.2) and (4) convolution layer 2 output (dark blue in figure IV.2) vectors as the input features of the four classical algorithms.

To better visualize the these high dimensional feature vectors, they are reduced into 3D using PCA and plotted in Figure IV.3, where each data point is colored by its true genre label. Features (1) and (2) are essentially the same under PCA transform, therefore only three plots are presented. We can make several observations from this figure.

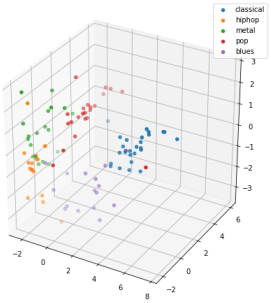
- In terms of raw flattened MFCC features, classical, blues, and pop music tend to stand alone by themselves, but hip-hop and metal are mixed together in the middle of the data points.
- In terms of DCNN layer 1 features, pop music separates itself away from the others on the right “branch”. Also, on the left branch, hip-hop music now is separated from metal music, although metal music now seem even more mixed up with all the other music genres.
- Although no dicision boundary can be seen seen directly from PCA of layer 2, all of the five genres has a clear



(a) Raw flattened MFCC vector



(b) DCNN Layer 1



(c) DCNN Layer 2

Figure IV.3: Three different feature vectors of the test set data points (music pieces), with the vectors reduced to 3D using PCA. The colors represent true genre labels of each test data point.

tendency of clustering. Moreover, compared with the other two features, the points are more condensed.

It is worth noticing that these observations are based only on a 3D reduction of these features. Points could be well separated in higher dimensions, even if the points seem not clearly separated apart in 3D.

V. Results and Discussion [2]

A. Dilated CNN Modeling

After some hyperparameter tuning, we decided to use the Adam optimizer with standard parameters and learning rate 0.001. Here we investigate whether the number of layers in the model will impact on the performance of the DCNN by comparing a 2-layer model and a 3-layer model. Both networks have trained with well selected parameters. We used drop off in both with rate 50% to reduce overfitting. The results are presented in Table V.1.

Model	Train Accuracy	Test Accuracy
2-layer Dilated CNN	0.915	0.87
3-layer Dilated CNN	0.9125	0.79

Table V.1: Different Model architecture's comparison

As the table shows, the training error of both 2-layer and 3-layer models are around 91%, but the difference between train and test accuracy for a 3-layer model is larger than that of a 2-layer model, suggesting that the 3-layer model suffers from overfit the data much more than 2-layer model.

[2] All experiments here are part of the repository https://github.com/LithiumH/CS229-Music-Classification

1) Filter size and Pool size: Here we investigated whether the filter size and pool size make a difference in our predicting performance of the model. We iterated on only 2-layer Dilated networks because they have the best test results from the previous part. The results are presented in Table V.2. The first two columns encode information about the two dilated convolution layers: the four numbers in each cell, separated by a comma, are the number of filters, size of filters, dilation rate, and pool size, respectively. The prefix "d" denotes that a drop off of 50% was applied to the input of the layer.

First filter	second filter	Train Accuracy	Test Accuracy
1, 8, 4, 32	16, 4, 2, 8	0.51	0.49
8, 8, 4, 32	16, 4, 2, 8	0.93	0.85
8, 8, 4, d16	16, 8, 2, d4	0.825	0.69
8, 16, 8, d32	24, 16, 2, d4	0.9575	0.84
8, 64, 8, d32	24, 16, 2, d4	0.9925	0.84
16, 64, 8, d32	64, 16, 2, d4	1	0.84

Table V.2: Comparison of different model architectures: filter sizes and pool sizes.

We noticed that increasing the number of filters, filter sizes, and average pool size definitely improves test accuracy. We suspect this is due to the fact that music features are consistent throughout multiple section of the music, and the higher number of filter and filter sizes allows us to better extract the feature.

We eventually settled on the best set of parameters of the the simplest model with 0.84 test accuracy since test accuracy start to flat out. The confusion matrix of the model is shown below:

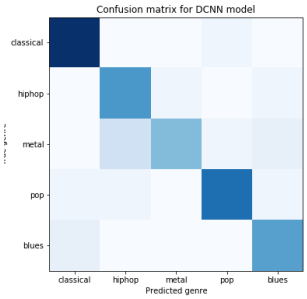


Figure V.1: DCNN confusion Matrix

Not surprisingly Metal is the hardest one to identify, while classical music can be identified with almost 100% accuracy

B. Classical Algorithms

In this section, we present the classification outcomes from the four classical algorithms, LR, GDA, random forest and SVM, with three different sets of input features: the baseline model using flattened MFCC matrix and combined models using two convoluted layer output features.

The final classification accuracy of our baseline model and the combined model can be found in Table V.3.

Using Table V.3, we make several observations.

Features (length) \ Classical Algs		LR	GDA	RF	SVM
Raw input (25800)	*train*	1.0	0.8875	1.0	1.0
	test	0.77	0.77	0.76	0.28
PCA reduces raw input (50)	*train*	0.925	0.84	1.0	1.0
	test	0.81	0.79	0.77	0.21
DCNN layer 1 (320)	*train*	0.99	1.0	0.99	0.92
	test	0.87	0.61	0.86	0.87
DCNN layer 2 (48)	*train*	0.95	0.94	0.98	0.94
	test	0.84	0.80	0.87	0.84

Table V.3: Comparison: Train and test accuracy of the four classical algorithms, using different feature inputs.

- The performance of GDA is very abnormal compared to the other algorithms. Using the raw input, it fails to achieve 100% accuracy on the training set while all the other three algorithms achieved this. We suspect this is because GDA makes strong underlying assumptions of the data to come from a Gaussian Distribution, while music temporal data generally does not follow a Gaussian Distribution. We also observe that after the feature extraction, GDA tend to perform better on the later layers of the CNN, which has a smaller number of activations. This is likely because GDA tend to perform better on low dimensional data. Also, that GDA works better in layer 2 than layer 1 corresponds to our observation from Figure IV.3(c), that the data points of each genre tend to cluster together in convolution layer 2.
- The performance of logistic regression and RF improved after PCA, and improved further with layer 1 and 2 of the Dilated CNN features. Interestingly, they out performed the DCNN, which was used to extract the features! We believe that there a few possible reasons. First, our neural network architecture might not yet be optimized for this problem, since there is still a gap between the training and test accuracy. Second, classical algorithms have much fewer parameters compared to the neural network, so they may actually have a regularizing effect on the activations of the neural network, thus helping us get higher test accuracy.
- The column of results for SVM is particularly interesting. We see that SVM performs extremely poorly when applied to the raw input. There is a huge gap between the train and test accuracy, which indicates severe overfitting. This is likely because the kernalized SVM has very strong predictive powers and will end up fitting very complex boundaries to the raw input in a high dimensional space that do not generalize to the test data. However, after PCA, SVM performs *even worse*. Compare this with the three other algorithms, whose performance *increases* after PCA. These results lead us to suspect that dimension reduction on the data does not help with the performance of SVM. However, when we use the layers of the DCNN as input, the performance of SVM *increases* dramatically. This indicates that the DCNN extracts low dimensional features from the raw input in a way that is fundamentally different from PCA. In particular, compare the results of PCA and DCNN layer 2. One has 50 features and the other 48 features, but the accuracy is 0.21 compared to 0.84.

VI. Visualization

We run PCA with true and predicted labels of Logistic Regression of the first layer features. The plots are shown in VI.1

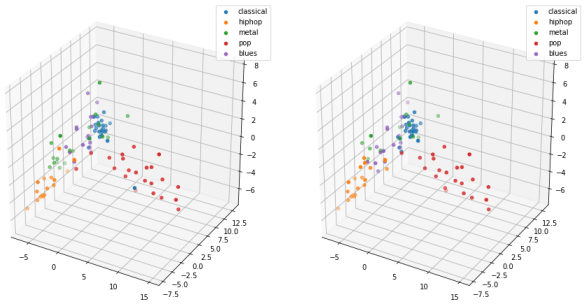


Figure VI.1: LR on First Layer of DCNN of test data with true (left) and predicted (right) genres

As we can see from the plots, there is a blue dot among a sea of reds, indicating that there was a classical music that had features (extracted from DCNN) that is very similar to pop music. Thus, it made sense for LR and other algorithms to classify it wrongly as a pop music as you can see on the figure on the right. We hypothesis that even though logistic regression have a slight regularization effect, it is ultimately constrained by how well the DCNN extract the features.

VII. Conclusions

We have successfully improved the accuracy of classical classification algorithms by using a Dilated Convolutional Neural Network as our feature extractor. In some cases, the performance of the classical algorithm can even exceed that of the neural network. In practice, this will allow us to use a pre-trained neural network as a feature extractor and improve both the performance and speed of classical algorithms. We have also made a few interesting observations regarding the results in Table V.3 and attempted to give an explanation. We are particularly impressed by the dramatic increase in performance when combining SVMs with the DCNN.

VIII. Future Work

There are several directions that would be interesting to pursue in the future. In our work we used a Dilated CNN as our feature extractor. However, since music data is inherently sequential, other network structures such as LSTM (long short term memory) and GRU (gated recurrent unit) will likely achieve better performance. Moreover, if we are able to train a much deeper network that attains high accuracy, it would be interesting to plot the accuracy of a classical algorithm using different activation layers. Additionally, we still do not have a full understanding of the results in Table V.3. It would require some theoretical work to explain the observations that we have made.

References

[1] Ioffe, Sergey and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." *ICML* (2015).

[2] Marsyas "Data Sets". G. Tzanetakis and P. Cook, *GTZAN Genre Collection*. http://marsyasweb.appspot.com/download/data_sets/

[3] Mutiara, A., Refianti, R. and Mukarromah, N. (2016). "Musical Genre Classification Using SVM and Audio Features." *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, 14(3), p.1024.

[4] T. LH. Li, A. B. Chan, and A. HW. Chun. "Automatic musical pattern feature extraction using convolutional neural network." *In Proc. of the Int. MultiConf. of Engi- neers and Computer Scientists (IMECS)*, Hong Kong, March 2010.

[5] Tzanetakis, G. and Cook, P. (2002). "Musical genre classification of audio signals." *IEEE Transactions on Speech and Audio Processing*, 10(5), pp.293-302.

[6] Lidy, Thomas, and Andreas Rauber. "Evaluation of feature extractors and psycho-acoustic transformations for music genre classification." *In ISMIR, pp. 34-41*. 2005.

[7] Ellis, Daniel PW. "Classifying Music Audio with Timbral and Chroma Features." In Ismir, vol. 7, pp. 339-340. 2007.

[8] Tsaptsinos, Alexandros. "Lyrics-based music genre classification using a hierarchical attention network." arXiv preprint arXiv:1707.04678 (2017).

[9] Karpov, Igor, and Devika Subramanian. "Hidden Markov classification for musical genres." Course Project (2002).

[10] Trevor, Hastie, Tibshirani Robert, and Friedman JH. "The elements of statistical learning: data mining, inference, and prediction." (2009).

IX. Contributions

- Haojun - Pre-processing. Dilated CNN modeling and experiments. Feature extraction. Drawing and generating pretty pictures.
- Jialun - Classical algorithms and experiments. Result inferences. Poster lead.
- Siqi - Classical algorithms and experiments. Report lead. Result discussions.