Music Classifier

Classifying Music according to Composers

Introduction

“ Where words fail, music speaks”- Hans Christian Andersen, that’s why music is so interesting and why People love to listen to music coming from specific person.

➔ **General Task**

Classify music tracks according to their composers.

➔ **Helpful**

It can be generalized to other music recognition tasks.

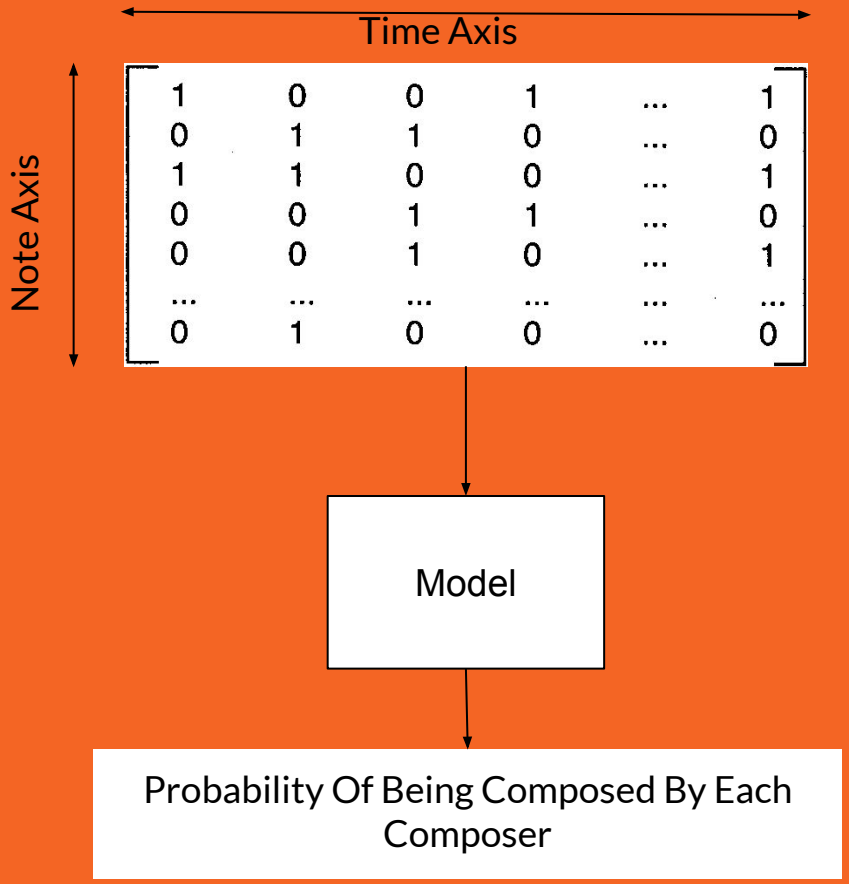
Input & Output

➔ **Input**

It is a midi file, which contains tracks and chords represented as a tensor with dimensions of #tracks, #notes, #time. Whether we'll sample from it or use it fully is something we will consider as a configuration problem since it would certainly affect performance. In our case, it is a 30 second music interval of 1 track with 128 notes.

➔ **Output**

The output is a prediction vector about which particular composer composed the music.


Time Axis
Note Axis
1 0 0 1 ... 1
0 1 1 0 ... 0
1 1 0 0 ... 1
0 0 1 1 ... 0
0 0 1 0 ... 1
... ...
0 1 0 0 ... 0
Model
Probability Of Being Composed By Each Composer

State Of The Art

CNN architecture, LSTM architectures and SVM as baseline.

This is the results from the stanford base paper:

Architecture	Train Examples	Val Examples	Test Exaxmples	Train Acc	Test Acc
Baseline	-	-	-	-	0.33
CNN	**~ 30,000**	**~ 300**	**~ 300**	**0.51**	**0.58**
LSTM 1	1414	313	354	0.2327	0.23
LSTM 2	1514	379	335	0.2325	0.26

Reference:

http://cs230.stanford.edu/projects_fall_2018/reports/12441334.pdf

Dataset

The dataset is 450 classical music scraped from the website provided below, and the dataset is already provided in their repository. This is the data for the 9 composers only.

- **Website Link:**
  http://www.midiworld.com/classic.htm
- **State of the art repository:**

  https://github.com/chramsey/CS230-Final-Project

Adding Dataset

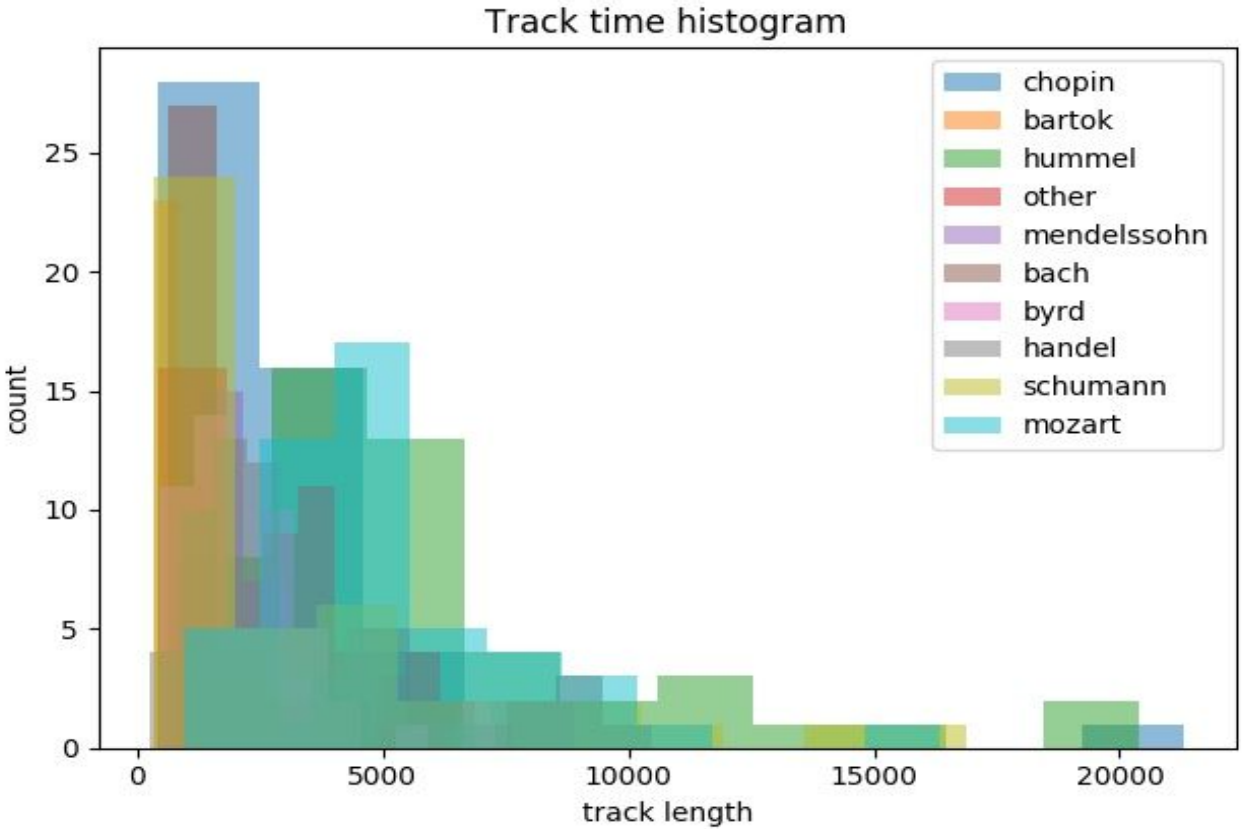
We obtained the original dataset from the original stanford report repository, which was a collection of 443 classical songs in midi formats for each of the 9 composers, with about 50 tracks per composer. However, we also added a collection of other midi files to represent the class of others in our project. We added 67 tracks to the dataset to represent our others class which included the work from composers such as Beethoven, Clementi and Haydn

Website Link of new Dataset:

http://www.piano-midi.de/

Exploring our Dataset

We explore here the frequency of lengths of all tracks in each class



Our Work

➔ **Full implementation of pipeline**

Complete implementation of the pipeline used for our project.

➔ **Added to the Dataset**

We added to the dataset the midi files needed for the others class.

➔ **Research & implementation of data augmentation**

We finished all data augmentation needed to get our dataset balanced.

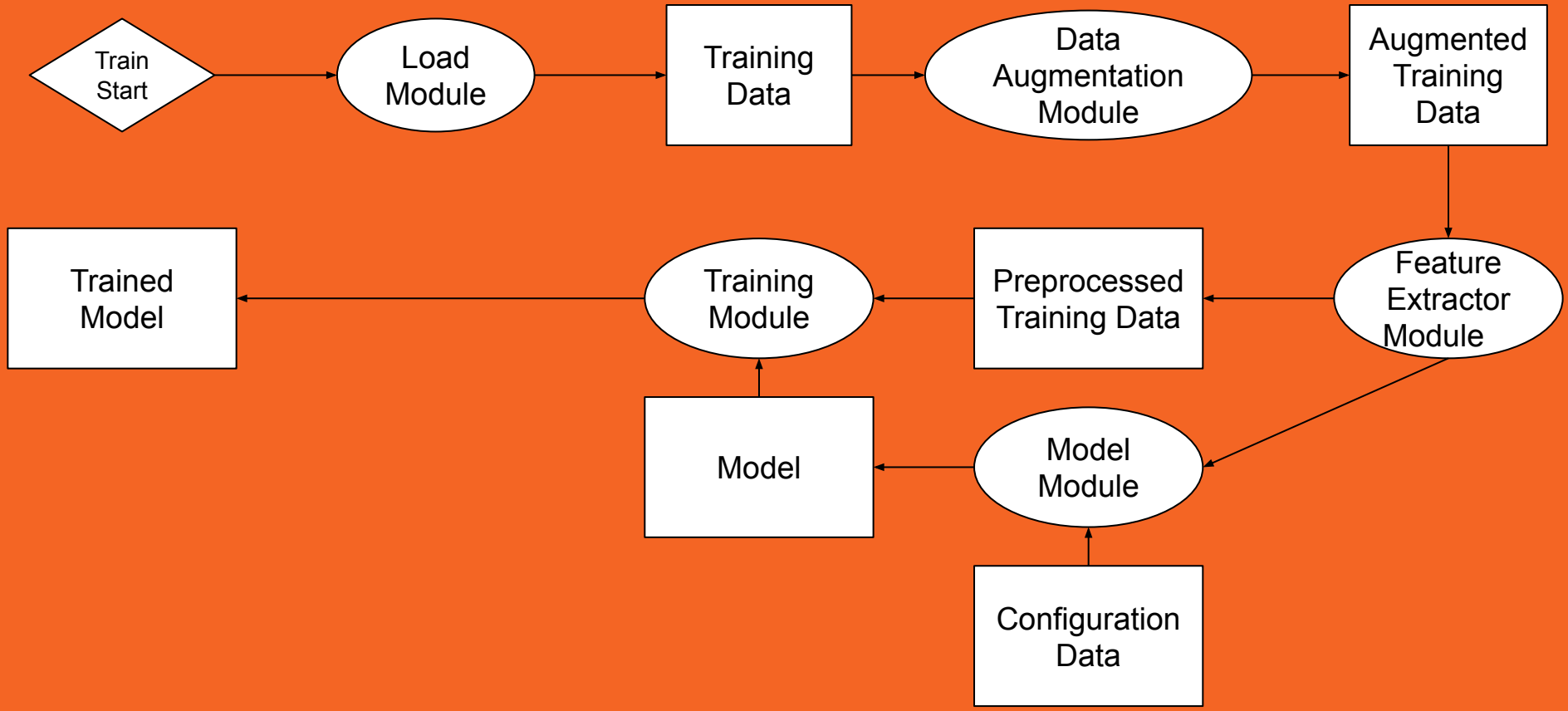
➔ **Research & implementation of feature extraction**

We finished all feature extraction needed to give our model extra features for classification.

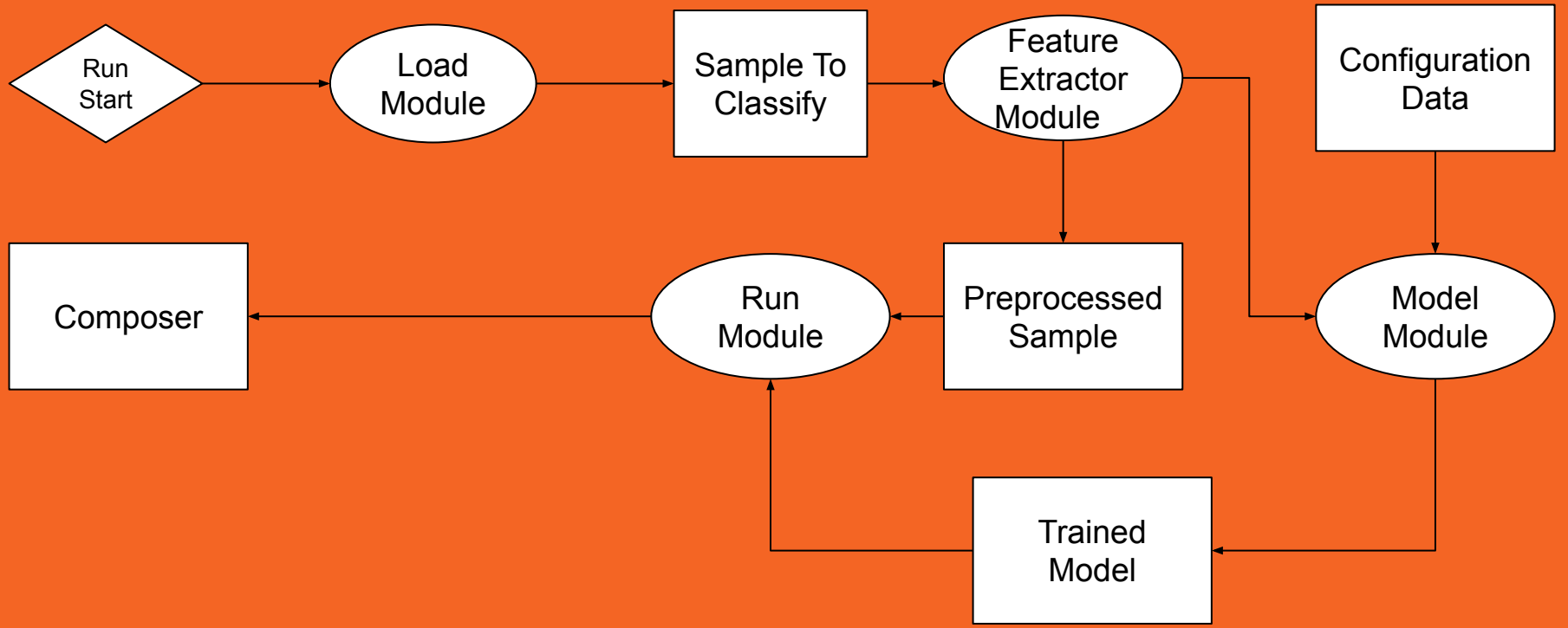
➔ **Build Models & Evaluation**

results for Cnn, Rnn and Hybrid Models

Our Project Architecture



Our Project Architecture



Research

Data Augmentation Research-cont.

➔**Learning Music Genres Article:**
(https://medium.com/@diegoagher/learning-music-genres-5ab1cabadfed)

In FMA dataset, each sample has a length of thirty seconds.
The trick to perform data augmentation is to split the thirty seconds into 10 chunks, 3 seconds each. Each window of the song will be tagged with the same genre of the original thirty seconds. If a song had rock as genre, then all 10 windows that came out of the splitting will have rock as genre. This trick gives 10x more data than the original dataset.

Data Augmentation Research

➔ **Justin Salamon & Juan Pablo Bello Paper** (https://arxiv.org/pdf/1608.04363.pdf)

It experiments 4 different data augmentations resulting in 5 augmentation sets:

1) Time Stretching:

Each sample is stretched by 4 factors.

2) Pitch Shifting:

Each sample is pitch shifted by 4 values (while duration is unchanged).

3) Dynamic Range Compression:

compress the dynamic range of the sample using 4 parameterizations { music standard, film standard, speech, radio}

4) Background Noise:

mix the sample with another recording containing background sounds from different types of acoustic scenes.
[{street-workers, street-traffic, street-people, park].

Feature Extraction Research-Cont

- **Predicting Hit Songs with MIDI Musical Features:**
  (http://cs229.stanford.edu/proj2014/Kedao%20Wang,%20Predicting%20Hit%20Songs%20with%20MIDI%20Musical%20Features.pdf)
  Features depends on Midi messages as : note on, note off, set tempo, program change and Delta time.
  1) Instrument: Each MIDI track contains a program change message, with instrument type encoded with 0 - 127.
  2) Melody: Melody features are represented by chord progression characteristics.
  - Consecutive two notes.
  - Consecutive Three notes

  3) Beats:
  - note duration and percussion

  4) Dimensionality reduction: PCA

Feature Extraction Research

➔ Music Genre Classification Article

(https://towardsdatascience.com/music-genre-classification-with-python-c714d032f0d8)

Using Librosa library

They extract **5** features:

1) Zero Crossing Rate

The rate at which the signal changes from positive to negative or back.

2) Spectral Centroid

It indicates where the centre of mass for a sound is located and is calculated as the weighted mean of the frequencies present in the sound.

3) Spectral Rolloff

It is a measure of the shape of the signal.

4) Mel-Frequency Cepstral Coefficients

It models the characteristics of the human voice.

5) Chroma Frequencies

Representation of the entire spectrum as it is projected onto 12 bins representing the 12 distinct semitones (or chroma) of the musical octave

Pipeline

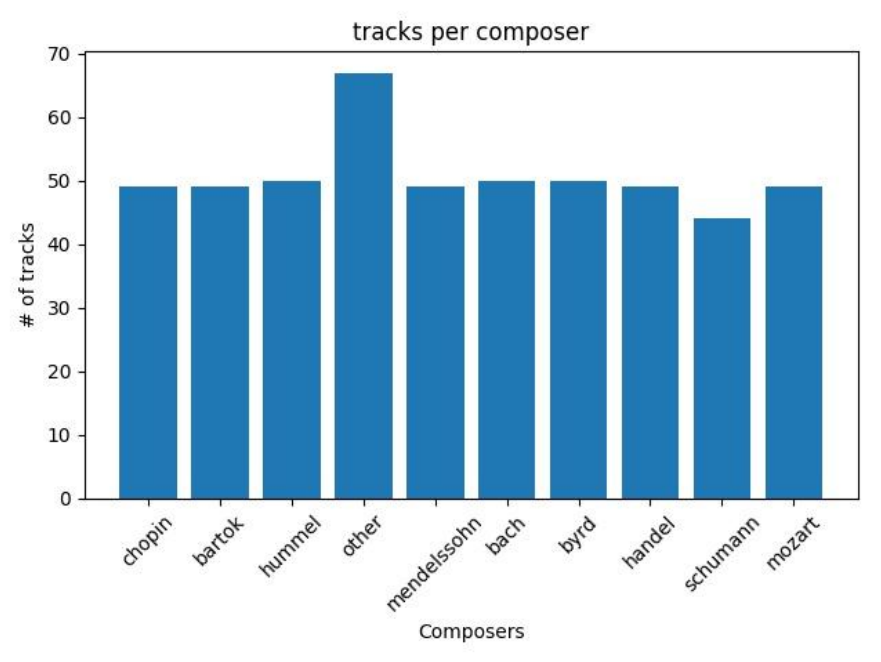
- **Setting up Configurations**
  Adjusts hyperParameters and running models.
- **PreProcessing,Augmentation and Balancing data**
  Adjusts dataset to get realistic results.
- **Output and Voting Strategy**
  Gets probability of being composed by each composer and predictions by voting or single test and shows the composer who take the highest number of votes.

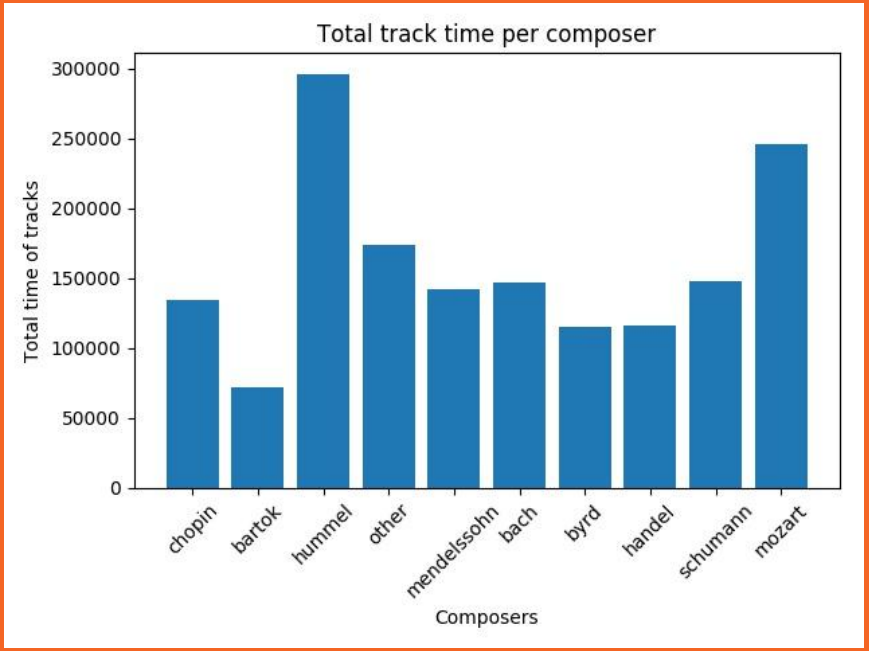
PreProcessing

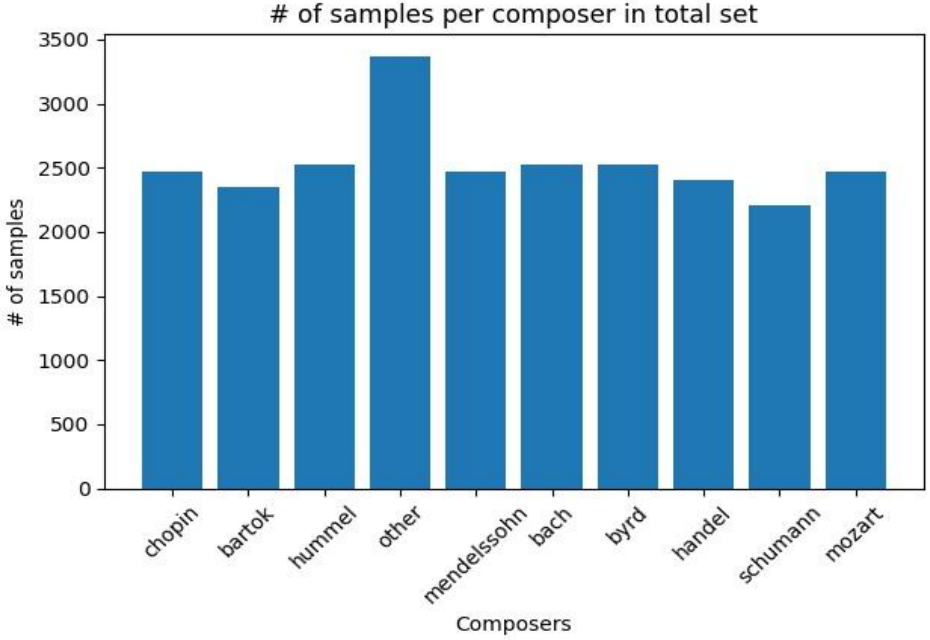
- We used the pretty midi python library to parse and act as a wrapper to the midi files.
- We extracted from each midi file its piano roll with a sample frequency of 10, the piano roll is a matrix representing the volume of each musical note across the different timesteps flattened across instruments.
- We filtered any tracks that are less than 30 second.
- We separated our dataset to training, validation and testing with 70%, 15% and 15% respectively.

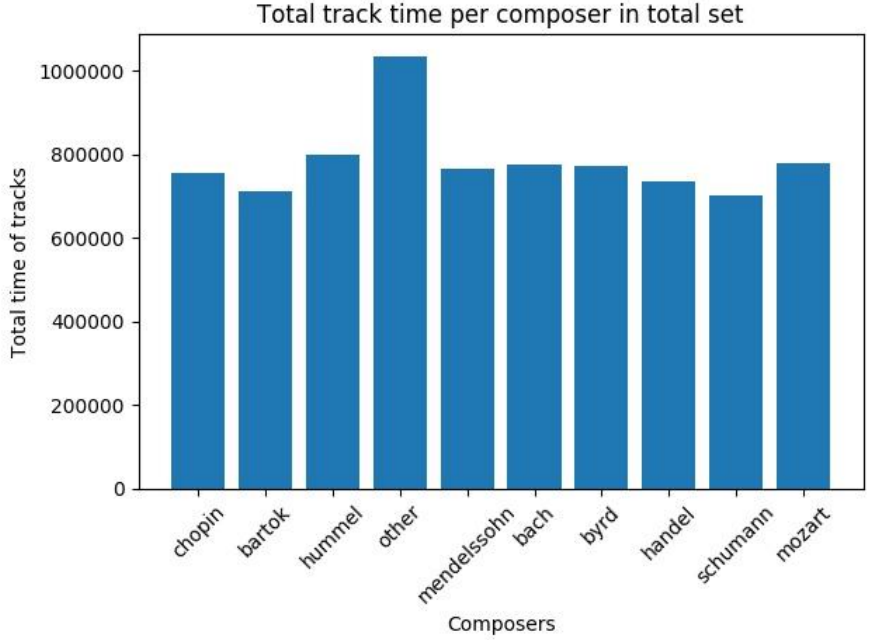
Data Augmentation & Balancing

- We augment the data by taking multiple windows of 30 second from each midi file we have and give it the label of the midi file.
- We balance the data by taking an equal amount of 30 second segments from each artist.






# of samples per composer in total set
# of samples
3500
3000
2500
2000
1500
1000
500
0
chopin
bartok
hummel
other
mendelssohn
bach
byrd
handel
schumann
mozart
Composers


Total track time per composer in total set
Total time of tracks
1000000
800000
600000
400000
200000
0
chopin
bartok
hummel
other
mendelssohn
bach
byrd
handel
schumann
mozart
Composers

Voting Strategy

Taking multiple windows of 30 second from the midi file we want to test and get prediction of model for it, then collecting votes from multiple predictions and select the composer who has the highest number of votes to be our prediction

This improves prediction of our model.

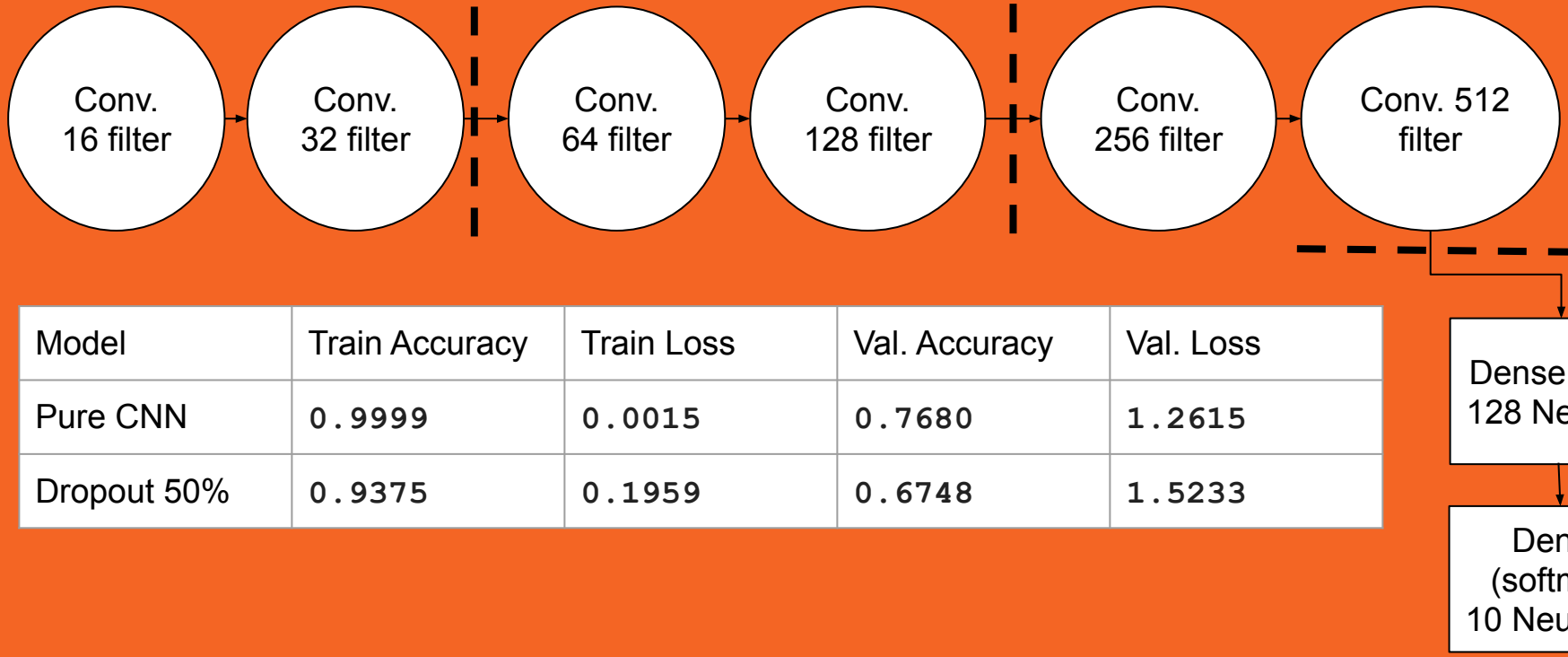
Results for models

Default Configurations

- Batch normalization was used after each layer
- If dropout was applied, it was applied on the dense layers only
- If L1 or L2 regularization was applied, it was applied on all layers
- Input to Dense layers is always flattened before it goes to it
- Adam optimizer with 0.001 learning rate, beta 1 with value 0.9 and beta 2 with value 0.999 were used as default optimizer

CNN

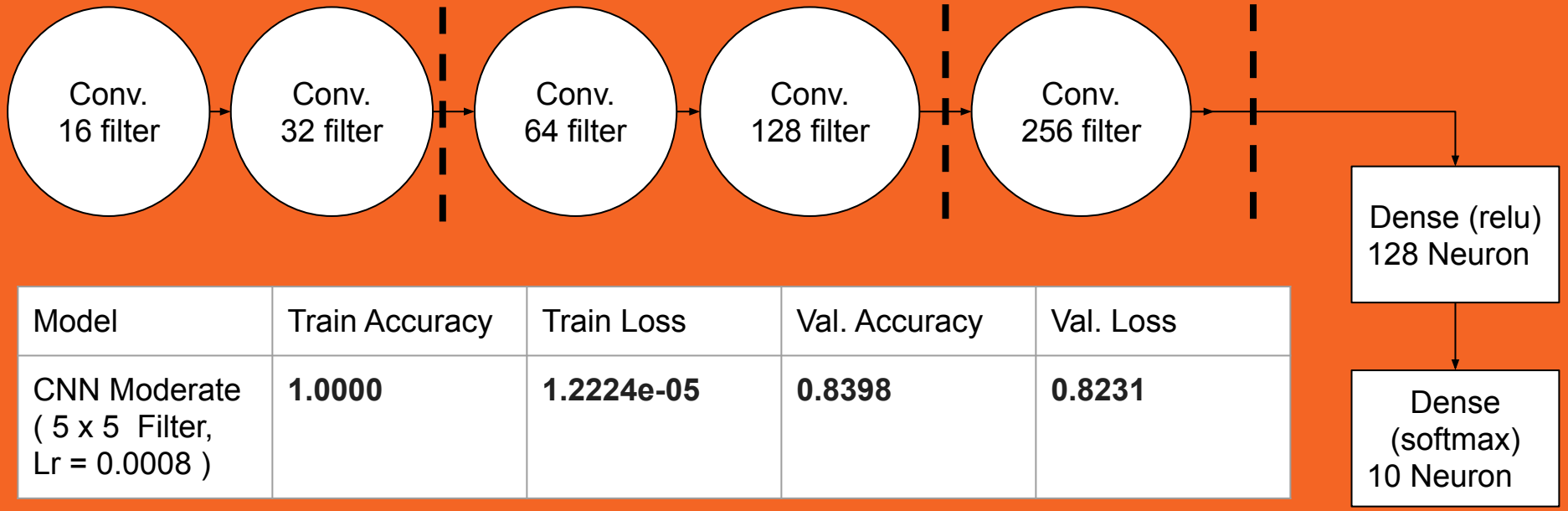
- All convolutional layers kernel sizes are 3x3.
- After each 2 convolutional layer, there is a max pooling layer.
- After each convolutional layer, there is a relu activation and batch normalization.



Model	Train Accuracy	Train Loss	Val. Accuracy	Val. Loss
Pure CNN	0.9999	0.0015	0.7680	1.2615
Dropout 50%	0.9375	0.1959	0.6748	1.5233

CNN Moderate

- All convolutional layers kernel sizes are 5x5.
- After each 2 convolutional layer, there is a max pooling layer.
- After each convolutional layer, there is a relu activation and batch normalization.



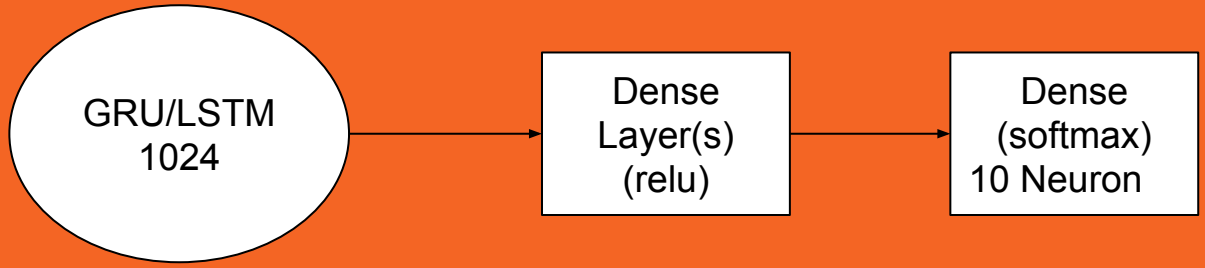
Model	Train Accuracy	Train Loss	Val. Accuracy	Val. Loss
CNN Moderate (5 x 5 Filter, Lr = 0.0008)	**1.0000**	**1.2224e-05**	**0.8398**	**0.8231**

Fine Tuning CNN

Model	Train Accuracy	Train Loss	Val. Accuracy	Val. Loss
Moderate_CNN (NoDropout,lr0.0008)	**1.0000**	**1.2224e-05**	**0.8398**	**0.8231**
Big_CNN_NoDropout (3x3, to 2048)	**0.9936**	**0.0233**	**0.7388**	**1.0968**
Moderate_CNN_L2 (3x3, penalty 0.001)	**0.3422**	**10266.6516**	**0.1664**	**326612.1782**
Moderate_CNN (No Dropout)	**1.0000**	**4.6925e-04**	**0.7750**	**1.0182**
Moderate_CNN (3x3, Dropout = 10%)	**0.9890**	**0.0405**	**0.5233**	**2.7380**
Moderate_CNN (Dropout = 10%)	**0.9818**	**0.0839**	**0.2481**	**5.2852**

RNN

- One LSTM or GRU with 1024 units.
- Dense layer(s).



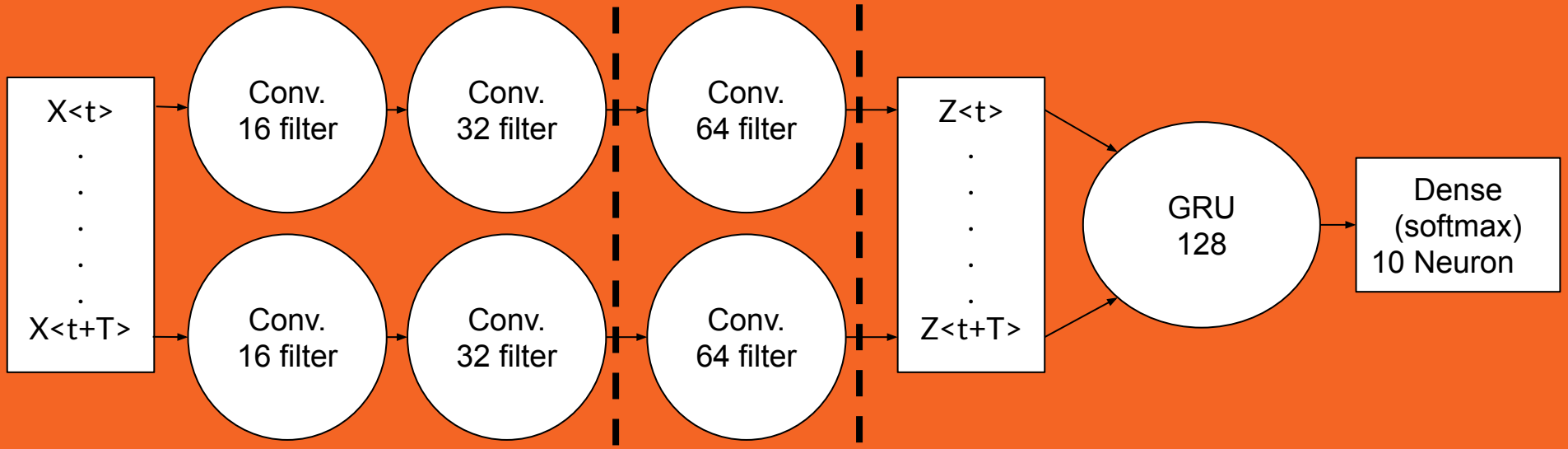
Model	Train Accuracy	Train Loss	Val. Accuracy	Val. Loss
Pure GRU RNN	**0.9949**	**0.0174**	**0.6136**	**2.3660**

Fine Tuning RNN

Model	Train Accuracy	Train Loss	Val. Accuracy	Val. Loss
RNN_GRU_Dense (one_128, No Dropout)	**0.9949**	**0.0174**	**0.6136**	**2.3660**
RNN_GRU_Dense (Three_512_128_64, No Dropout)	**0.9837**	**0.0521**	**0.6218**	**2.0767**
RNN_GRU_Dense (Two_256_128, No Dropout)	**0.9944**	**0.0207**	**0.6218**	**1.7632**
RNN_LSTM_Dense (Two_256_128, No Dropout)	**0.9723**	**0.0789**	**0.6120**	**2.1595**
RNN_GRU_Dense (one_128, Dropout = 10%)	**0.9719**	**0.0841**	**0.6831**	**1.6974**
RNN_GRU_Dense (Three512_128_64, Dropout=20%)	**0.9115**	**0.2709**	**0.6802**	**1.3123**

Hybrid

- All convolutional layers kernel sizes are 5x5.
- Each dotted line represents a max pooling layer
- After each convolutional layer, there is a relu activation and batch normalization.
- All convolutions are 1D, the convolutional model has parameter tying and is the same that is applied on all timesteps of input.



Fine Tuning Hybrid

Model	Train Accuracy	Train Loss	Val. Accuracy	Val. Loss
Hybrid_D0.7_L20.1_LR0.0004	**0.8509**	**0.4406**	**0.7386**	**0.7659**
Hybrid_D0.5_L20.01_LR0.0006	**0.9784**	**0.0791**	**0.7455**	**0.9403**
Hybrid_D0.5_L20.01_LR0.0007	**0.9425**	**0.1886**	**0.7383**	**0.8355**
Hybrid_D0.3_L20.002_LR0.0001	**0.9501**	**0.1787**	**0.7543**	**0.8170**
Hybrid_D0.2_L20.001_LR0.0007	**0.9286**	**0.2396**	**0.7388**	**0.7876**
Hybrid_D0.1_L20.0001	**0.9742**	**0.0955**	**0.7081**	**0.9987**
Hybrid	**0.9819**	**0.0709**	**0.7340**	**1.0648**
Conv1D16_GRU128_Drop0.25	**0.9401**	**0.1932**	**0.6004**	**1.4428**

Model	Simple Model ('None')	Best CNN Model	Best RNN Model	Best Hybrid Model
Test Loss (Segment)	12.670327808533246	0.9002239916546227	1.5467208797829863	0.8626234033447513
Test Accuracy (Segment)	0.20684005732959151	0.8251186	0.5957709	0.731491
Test Loss (Voting)	6.7333126	0.6482172	0.98850596	1.182173
Test Accuracy (Voting)	0.2682927	0.9268293	0.74390244	0.81707317

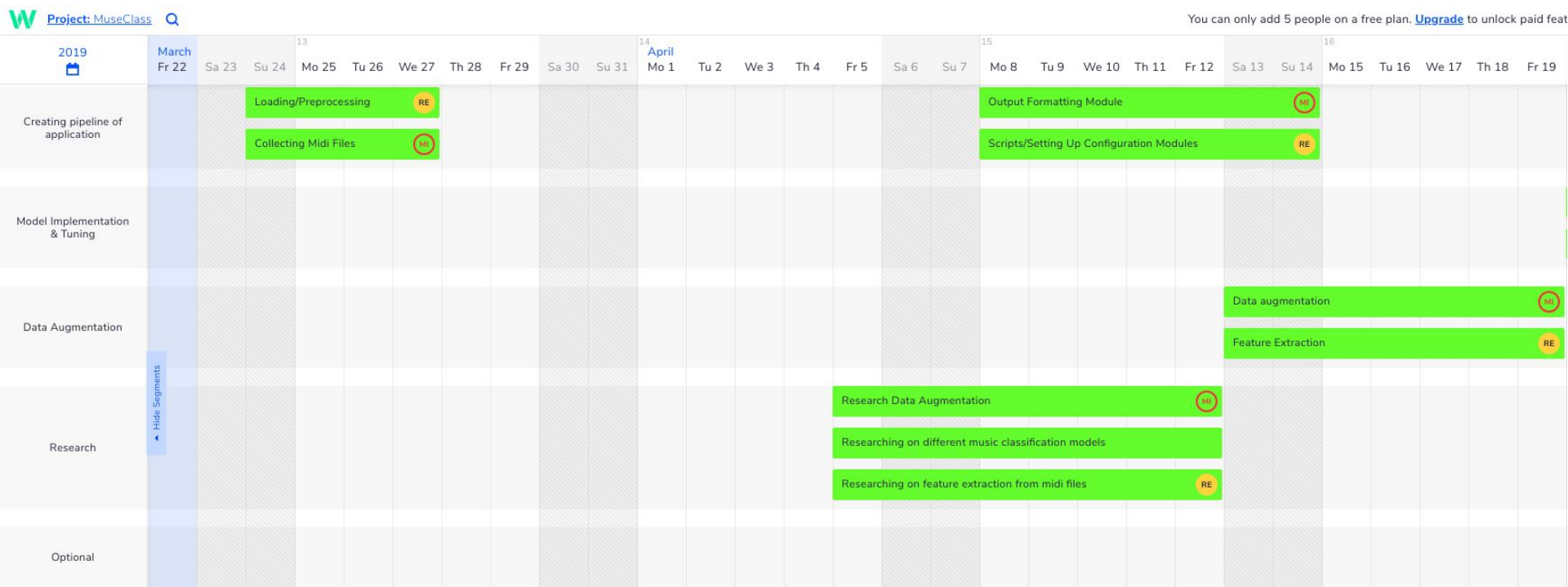
Results

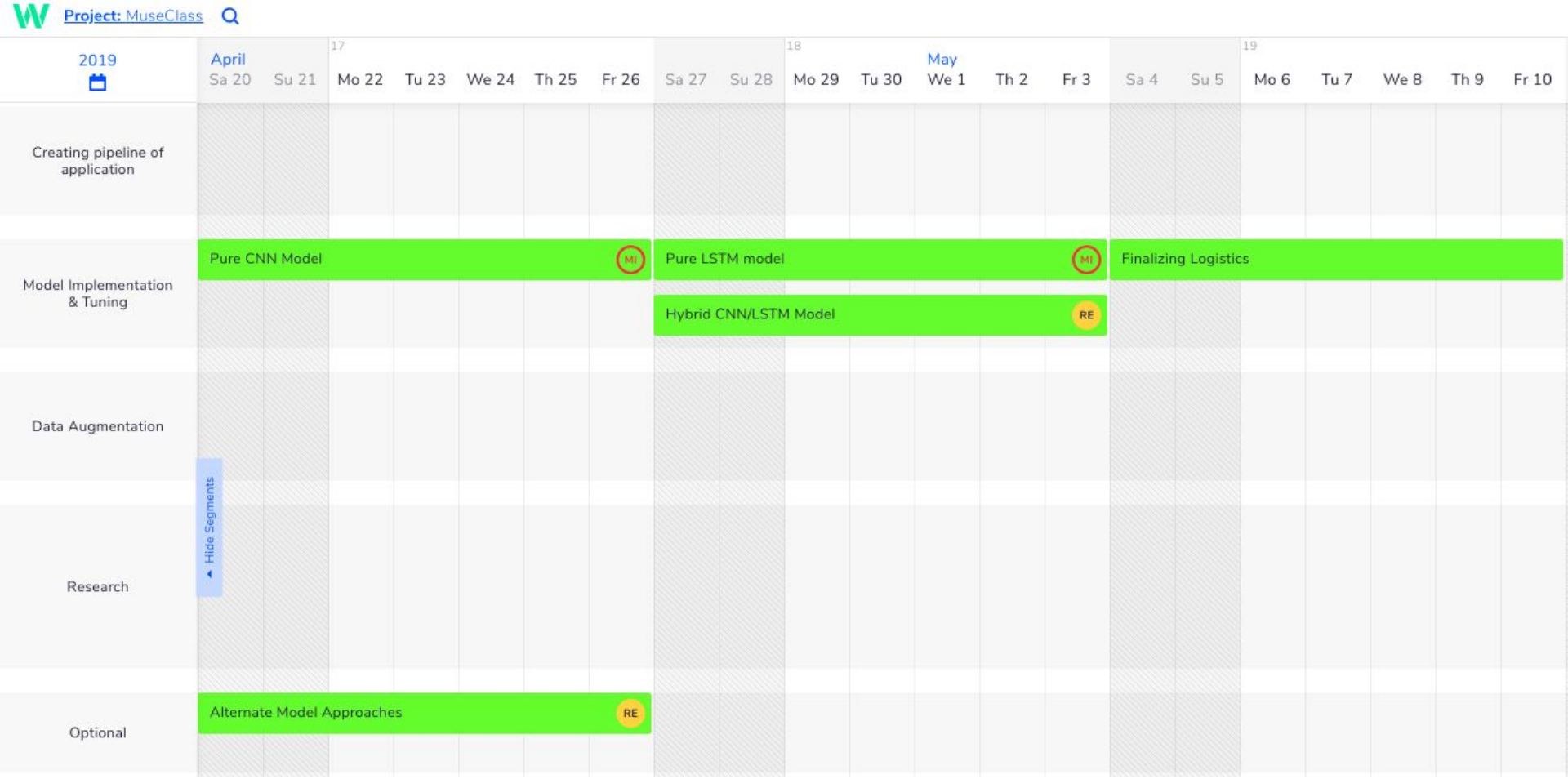
- **CNN Model**
  - 82.51% Single Accuracy
  - 92.68% Voting Accuracy
- **RNN Model**
  - 59.58% Single Accuracy
  - 74.39% Voting Accuracy
- **Hybrid Model**
  - 73.15% Single Accuracy
  - 81.70% Voting Accuracy

Gantt Chart Review

Gantt Chart & Task Assignment

We followed our Gantt Chart from the start till the end, it managed our time for every task.




Project: MuseClass
2019
April
May
Sa 20
Su 21
17
Mo 22
Tu 23
We 24
Th 25
Fr 26
Sa 27
Su 28
18
Mo 29
Tu 30
We 1
Th 2
Fr 3
Sa 4
Su 5
19
Mo 6
Tu 7
We 8
Th 9
Fr 10
Creating pipeline of application
Model Implementation & Tuning
Data Augmentation
Research
Optional
Pure CNN Model
MI
Pure LSTM model
MI
Finalizing Logistics
Hybrid CNN/LSTM Model
RE
Hide Segments
Alternate Model Approaches
RE

Challenges

- **Insufficient Hardware**
  Due to colab constraints, we were limited in our models especially hybrid ones since it ran out of memory regularly.
- **Unbalanced Dataset**
  Addressed by data augmentation.
- **Different Lengths Of Music**
  Addressed by windowing technique.

Next Steps

- **Applying Siamese network and one shot learning**

  Since we can notice that we can classify successfully over a few defined composers, this mean that each composer has a somewhat unique audio fingerprint. Then our focus should be shifted on generalizing this classification task by applying a siamese network approach to generalize our model.

- **Change to weighted voting**

  Allow weighted voting by utilizing the probabilities produced by the model as a weight for each vote instead of the current approach giving all votes equal weights.

Thank you