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.

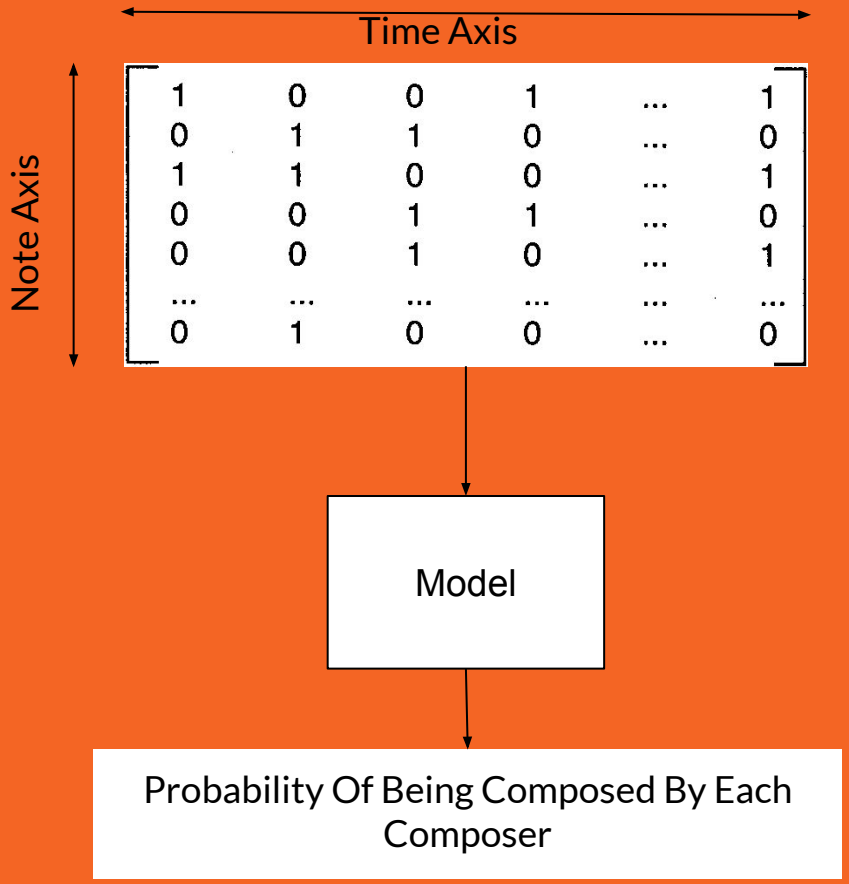
Input & Output

➔ **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

Pipeline

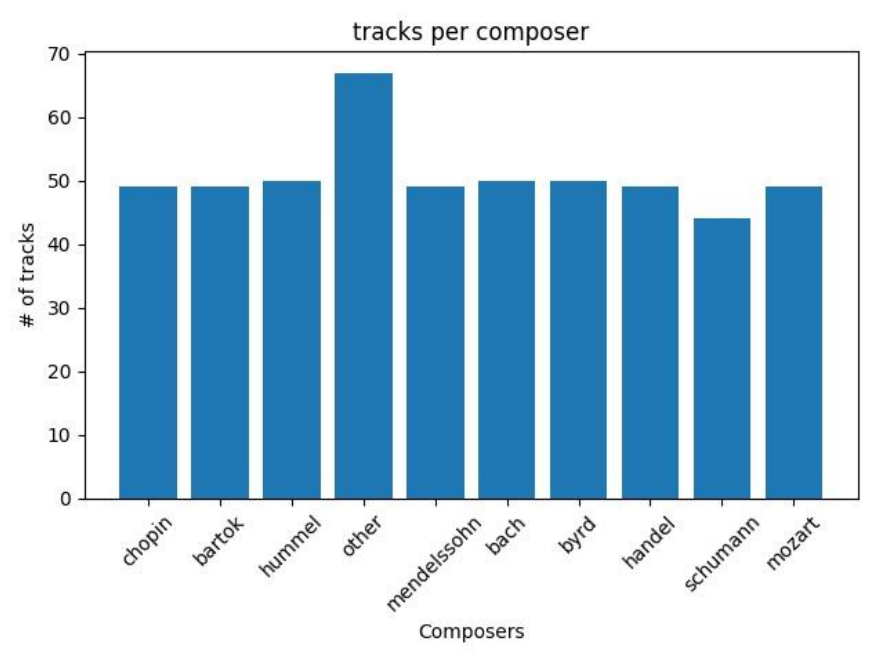
- **Setting up Configurations**
  Adjusts hyperParameters and running models.
- **PreProcessing,Augmentation and Balancing data**
  Adjusts dataset to get realistic results.
- **Output and Voting Strategy**
  Gets probability of being composed by each composer and predictions by voting or single test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

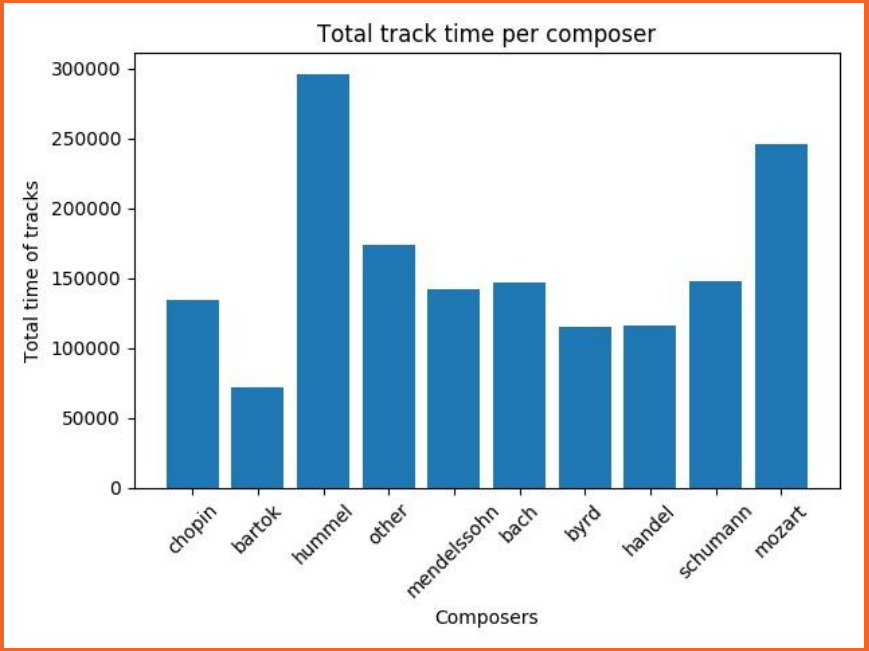
PreProcessing

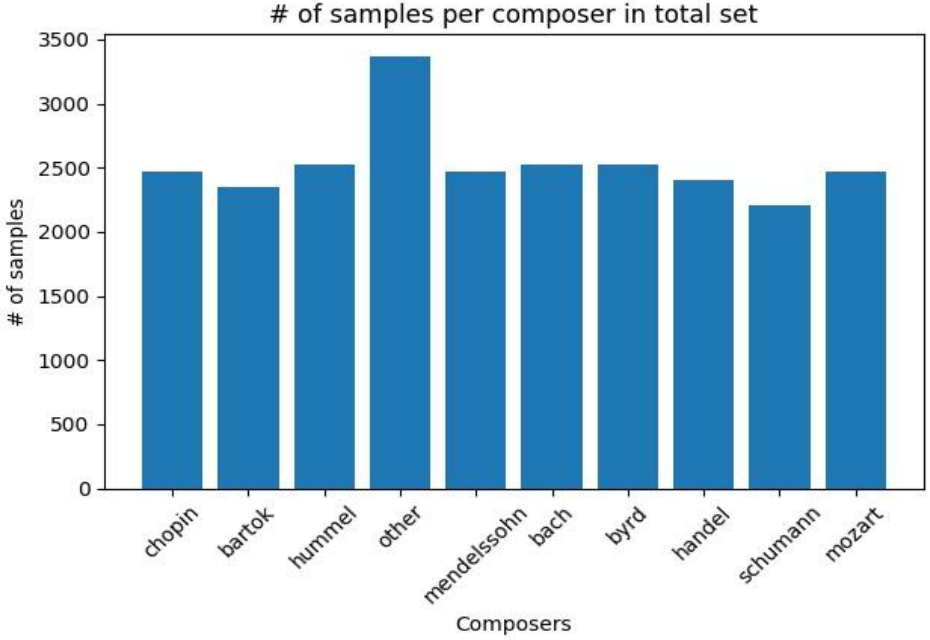
- We used the pretty midi python library to parse and act as a wrapper to the midi files.
- We extracted from each midi file its piano roll with a sample frequency of 10, the piano roll is a matrix representing the volume of each musical note across the different timesteps flattened across instruments.
- 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.

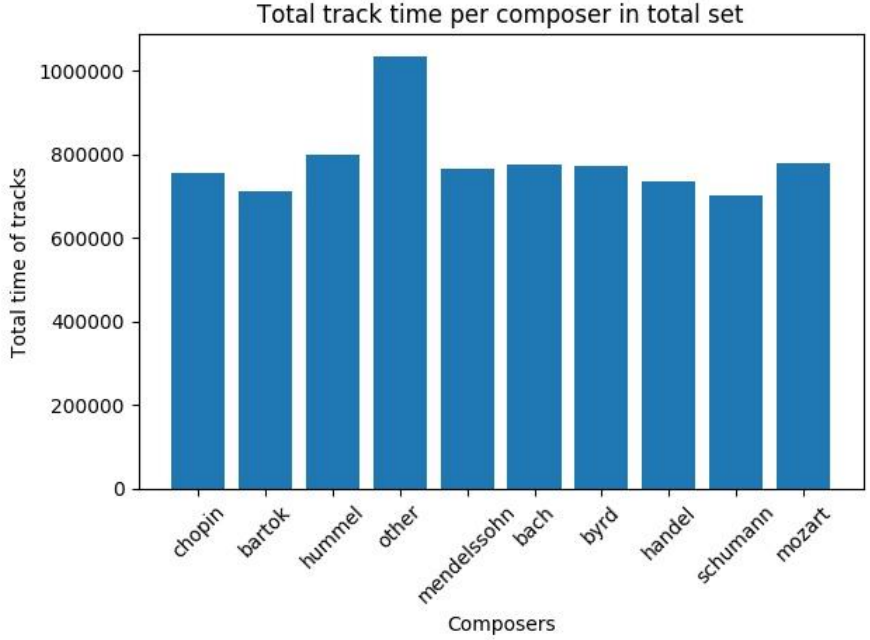
Data Augmentation & Balancing

- 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

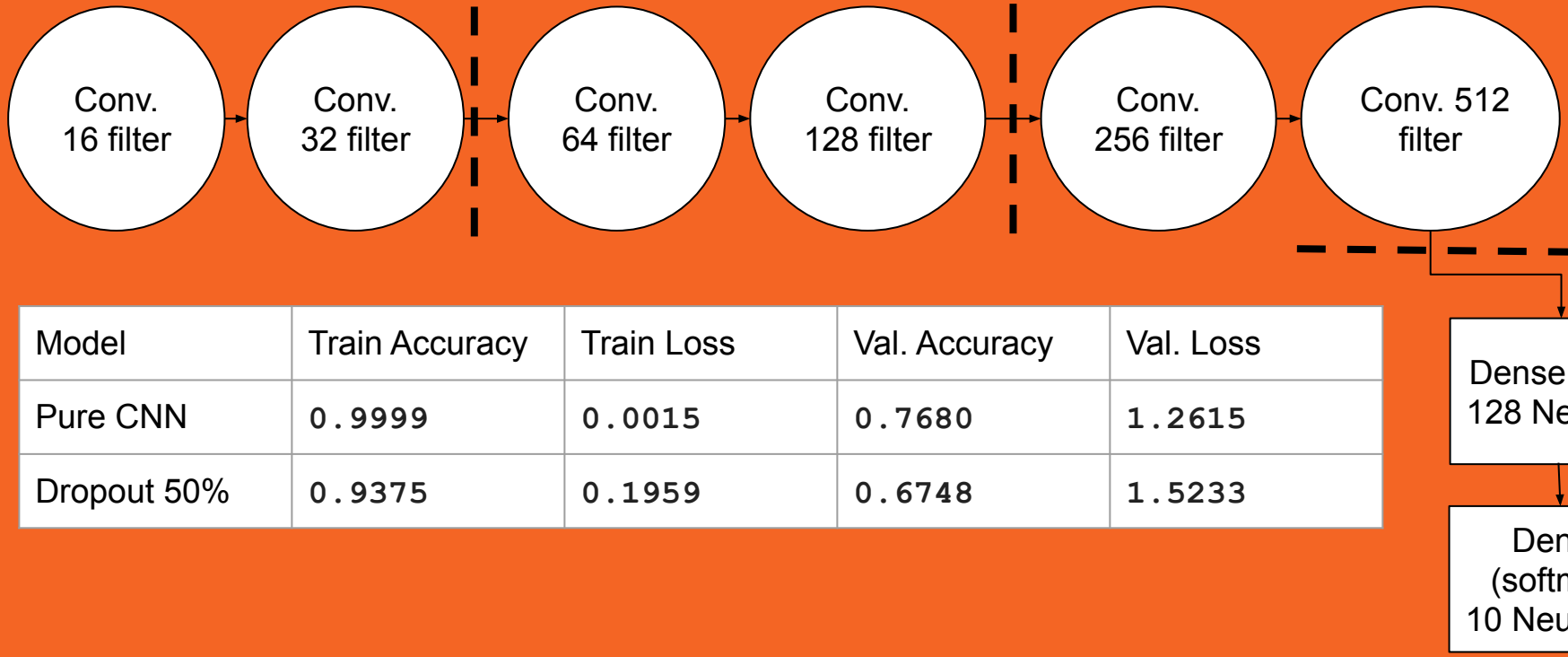
Results for models

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

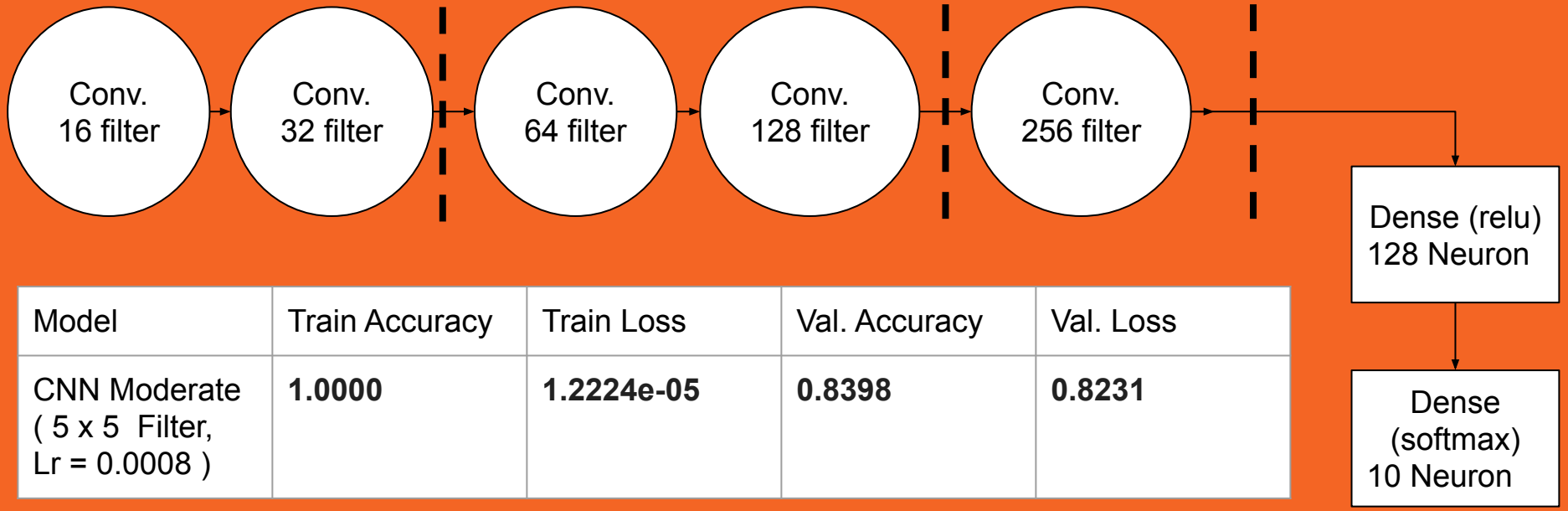
- All convolutional layers kernel sizes are 3x3.
- After each 2 convolutional layer, there is a max pooling layer.
- After each convolutional layer, there is a relu activation and batch normalization.



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.



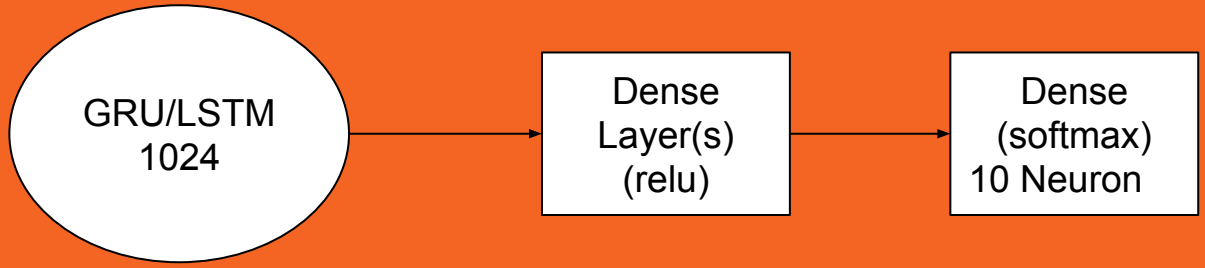
Model	Train Accuracy	Train Loss	Val. Accuracy	Val. Loss
CNN Moderate (5 x 5 Filter, Lr = 0.0008)	**1.0000**	**1.2224e-05**	**0.8398**	**0.8231**

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).



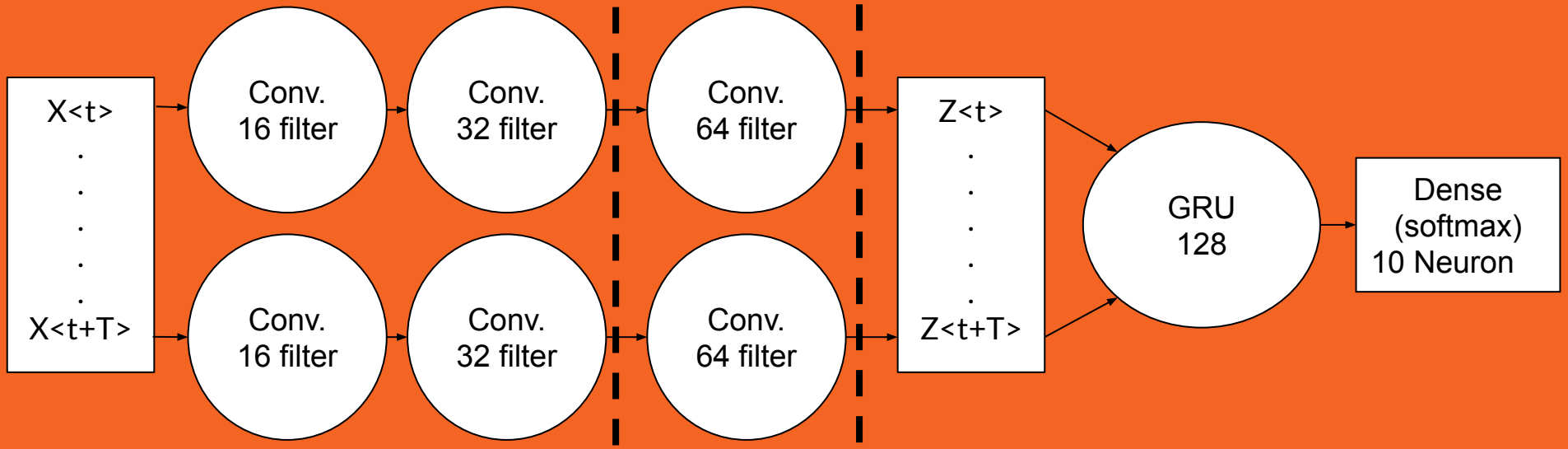
Model	Train Accuracy	Train Loss	Val. Accuracy	Val. Loss
Pure GRU RNN	**0.9949**	**0.0174**	**0.6136**	**2.3660**

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**

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.

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

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