MusicNet

Ziv Morgan
ID: 209904606
Github: https://github.com/zivm20/MusicNet-project/blob/main/project.ipynb

Dataset

The MusicNet dataset contains a collection of 330 classical music recordings, MIDI files and over 1 million labels indicating the precise time each note is played. For this project, I will be trying to classify both the composer of each recording, and the instruments being played.

The by reading the MIDI files we can extract the instruments played in the recordings and the composer can be found in the metadata.

This dataset proved quite challenging to work with, the lengths of the recordings are very different, the number of different samples is very low, and each recording has a very high sampling rate of 44100. Moreover, looking into the dataset a bit more, the dataset contains very high variance in the number of features. Below is the distribution of classes in the training dataset.

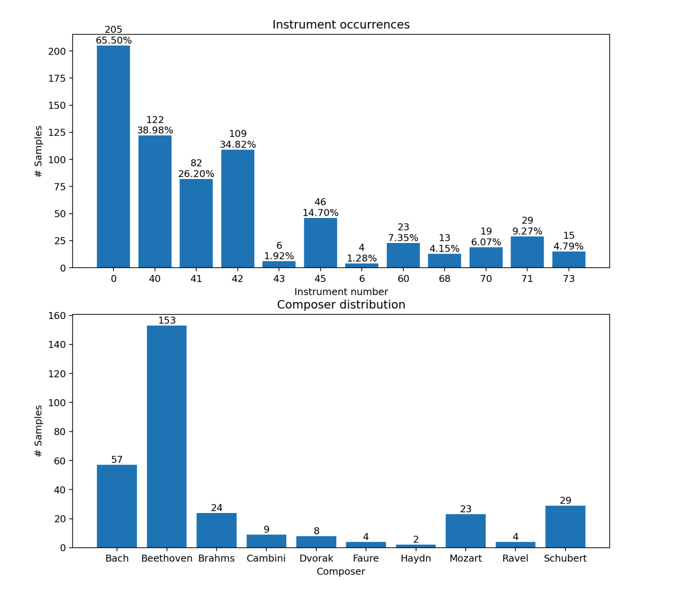


Figure 1: Distribution of initial training labels

To solve the high variance issue, any sample that contained an instrument that was played less than 10 times, and any sample that was played by a composer with less than 5 recordings was deleted. then a hard limit was set such that there would be only up to 85 samples that share the same composer. And lastly, samples with composers that have less then 85 recordings were duplicated until the number of samples was 85. Here are the training and validation datasets after the preprocessing (Note, no samples were removed from the validation set)

Figure 2: Distribution of training and validation labels after preprocessing

Doing this however made it so that the training set would have only 167 unique samples. In order to artificially introduce more samples into the dataset, each time a sample is loaded, only a 5 second window is randomly sampled from the actual audio file.

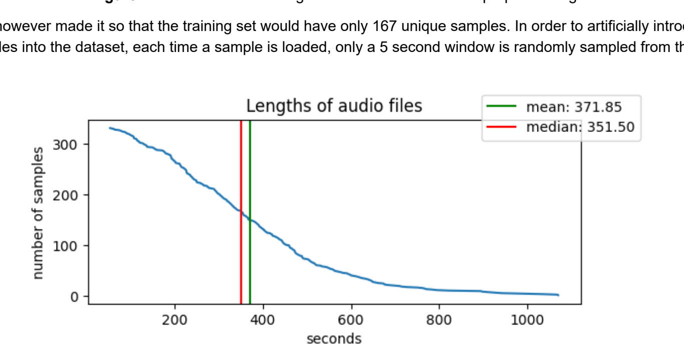


Figure 3: Lengths of all audio files

Therefore, giving us about $62,100$ training different samples in the dataset. Balancing out the dataset only ensures that at every epoch we will be training on the same number of composers.
However it is important to note that 2 samples from the same audio file might be very similar to each other.
At a sampling rate of 44100 we still get 220500 features per sample, so the audio files will be read using a sampling rate of 8000 and then saved on the disk.

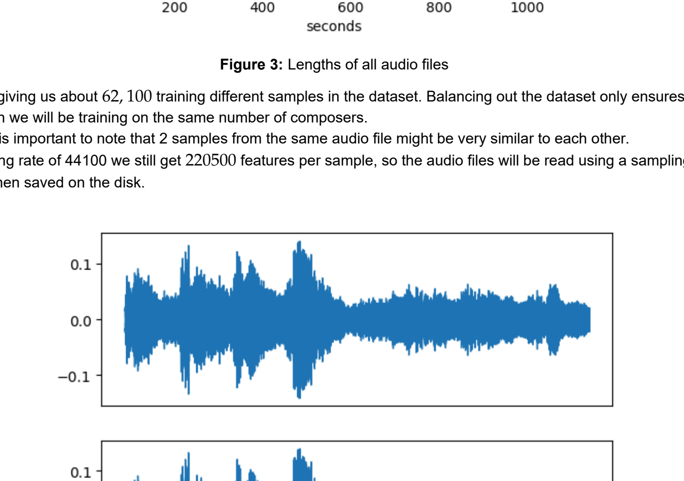


Figure 4: plot above has sampling rate of 44100, below only has 8000

Offline models

These models were only trained to classify composers as even that task was too much for them to handle.
Initially, I was going to try using offline models like KNN and AdaBoost with logistic regression as an estimator. KNN had a very fast run time compared to AdaBoost but it's performance was really bad, both on the train and validation set. Below is the performance of KNN during training with 595 samples

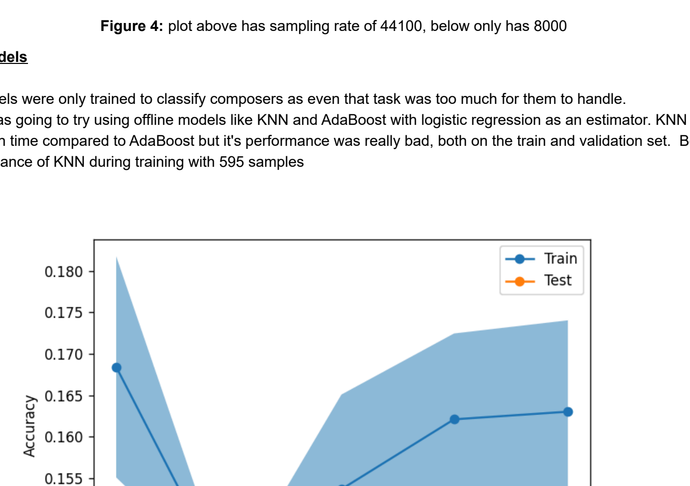


Figure 5: KNN model learning curve

The performance of KNN was poor due to the high dimensionality of the feature space.

As for the AdaBoost model, it managed to get a very good training accuracy but it was obviously overfitting as the number of samples was much lower than the number of features

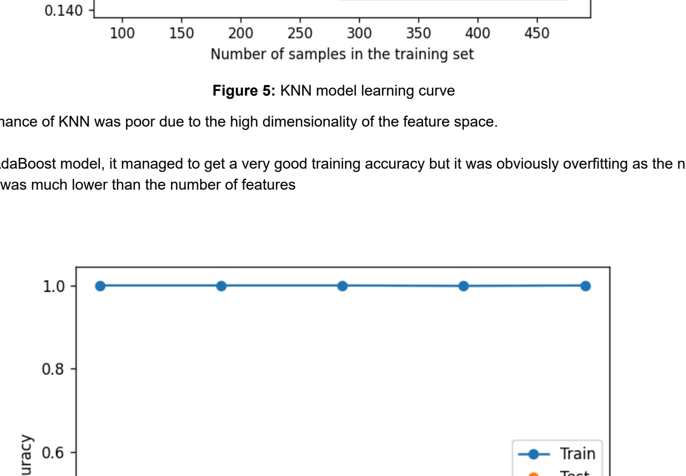
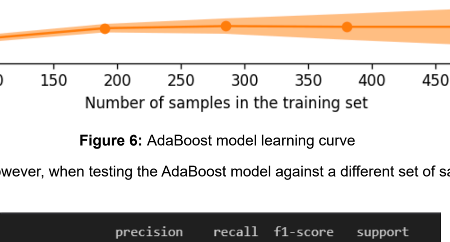


Figure 6: AdaBoost model learning curve

An interesting observation however, when testing the AdaBoost model against a different set of samples from the same training data,



	precision	recall	f1-score	support
0	0.00	0.00	0.00	85
1	0.26	0.14	0.18	85
2	0.21	0.08	0.12	85
3	0.21	0.06	0.09	85
4	0.30	0.04	0.06	85
5	0.25	0.06	0.10	85
6	0.15	0.79	0.25	85
accuracy			0.17	595
macro avg	0.20	0.17	0.11	595
weighted avg	0.20	0.17	0.11	595

Figure 7: classification report of the AdaBoost model on a regenerated train set

Using PCA we can attempt to reduce the number of features. That being said, we only have a limited amount of memory so we can only use a small portion of the dataset. Bellow is the explained variance of only 5950 samples.

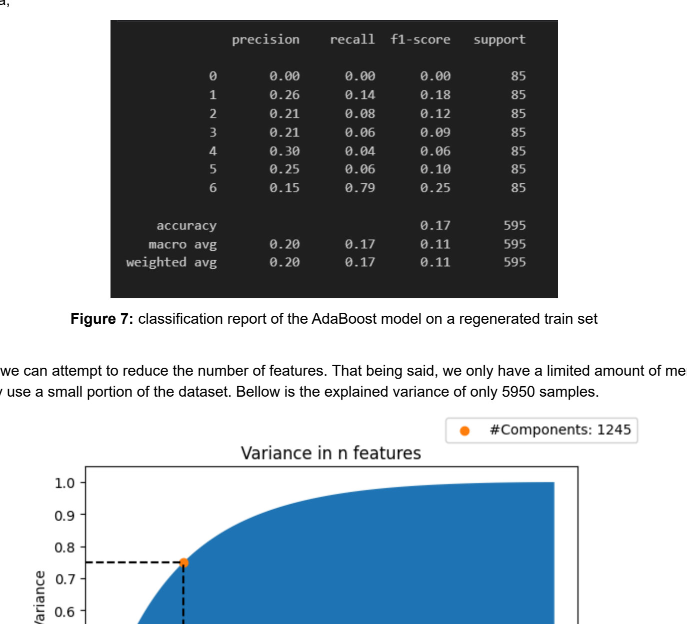


Figure 8: Explained variance from samples using PCA

to keep 75% of the information, we can reduce the number of features significantly. However, even with only 75% of the information, over 1000 features are needed.
Now that the number of features is much lower, the KNN model's performance was better than pure guesswork on the train set, but still lacks the complexity to find a general solution

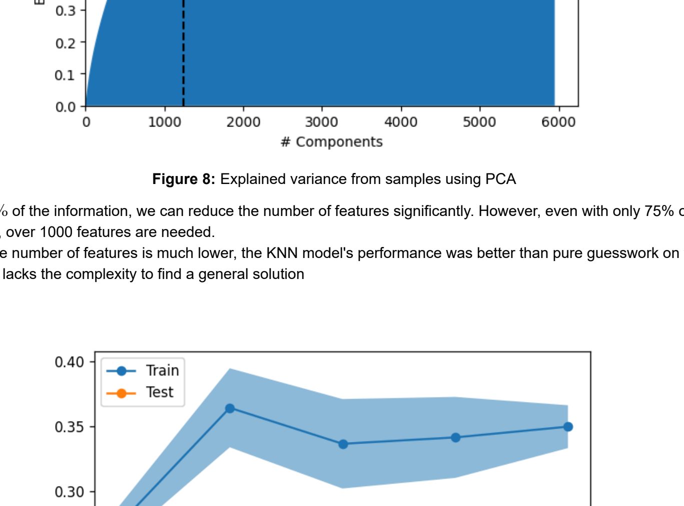


Figure 9: KNN model learning curve

The AdaBoost model also didn't manage to improve at all despite the reduction, the only improvement was a much faster run time

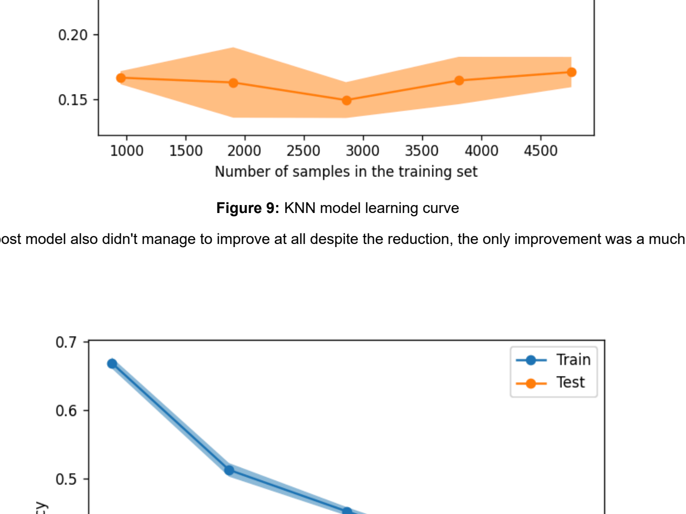


Figure 10: AdaBoost model learning curve

Perhaps by using more preprocessing steps and transformations to the audio we could solve the memory issue and perhaps find a better offline model. Or alternatively, we could just use an incremental model and thereby eliminate the issue completely.

Online models

When there is a lot of data to load the obvious solution is an online model that can train incrementally. In this section, a total of 3 different SGD classifiers were used, each with different loss functions, and finally a CNN. Additionally I tried adding some transformations to the data.

The SGD classifiers were as follows

Composer	Instrument
SGD classifier using Perceptron	SGD classifier using Perceptron
SGD classifier using hinge (soft margin)	SGD classifier using log loss
SGD classifier using modified huber loss	SGD classifier using modified huber loss

Were hinge loss is equivalent to Support Vector Classification $L(y_i, f(x_i)) = max(0, 1 - y_i f(x_i))$

and modified huber loss is $L(y_i, f(x_i)) = \begin{cases} max(0, 1 - y_i f(x_i))^2 & y_i f(x_i) > -1 \\ -4 y_i f(x_i) & y_i f(x_i) \le -1 \end{cases}$

Firstly, I tried training the 3 SGD classifiers on the dataset as is, the performance of this model was quite bad but it was clear that it was not overfitting, as in every epoch the dataset would be resampled randomly.

I tried a different approach, transforming the audio samples into a spectrogram thereby reducing the problem to an image classification problem

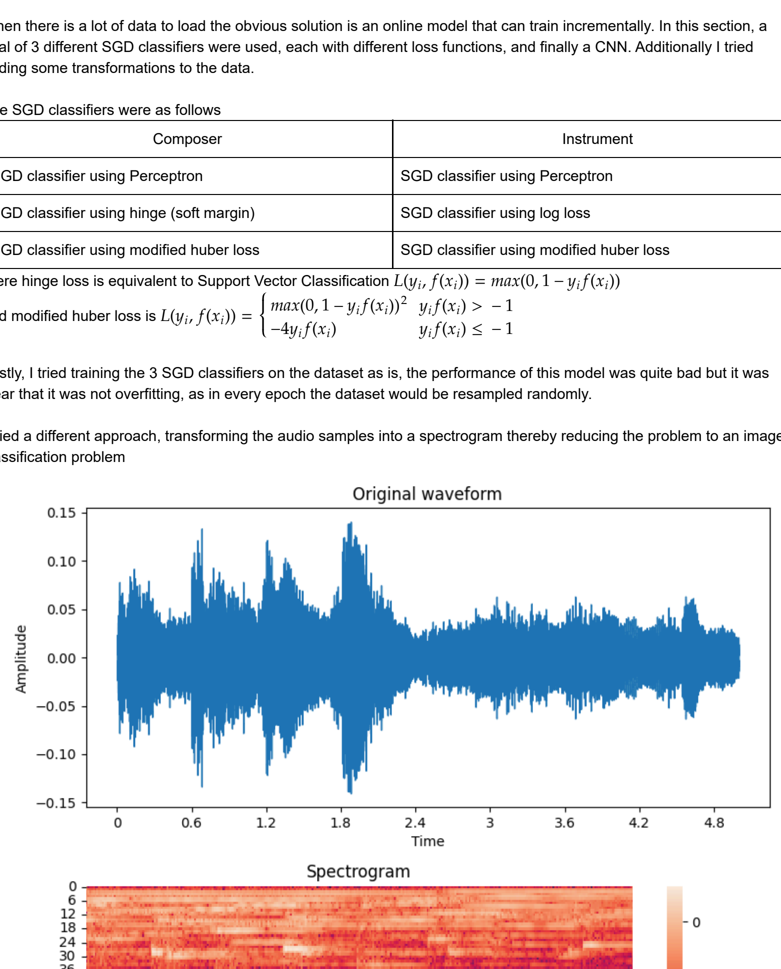


Figure 11: Spectrogram transformation

This transformation alone had a great impact on all 3 models performance for both composer and instrument classifications.

Another way to try and improve the model is to add more than one sample at a time, if the model could get multiple samples from the same audio file, there is a higher chance to get a sample that may contain clearer information, what if an instrument isn't played in the 5 second window?

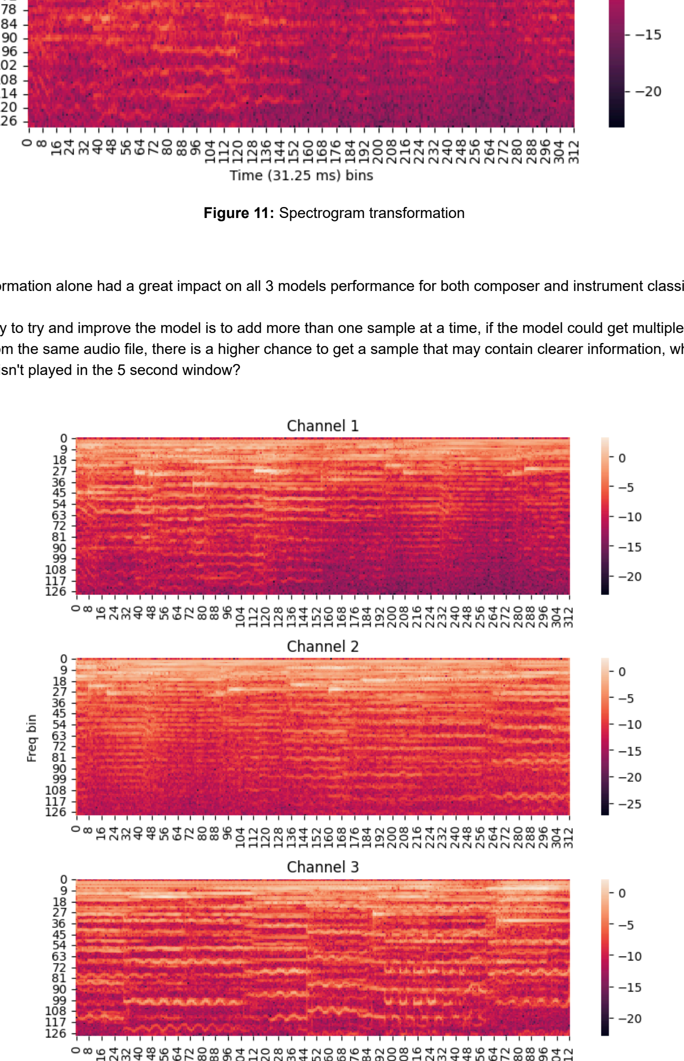


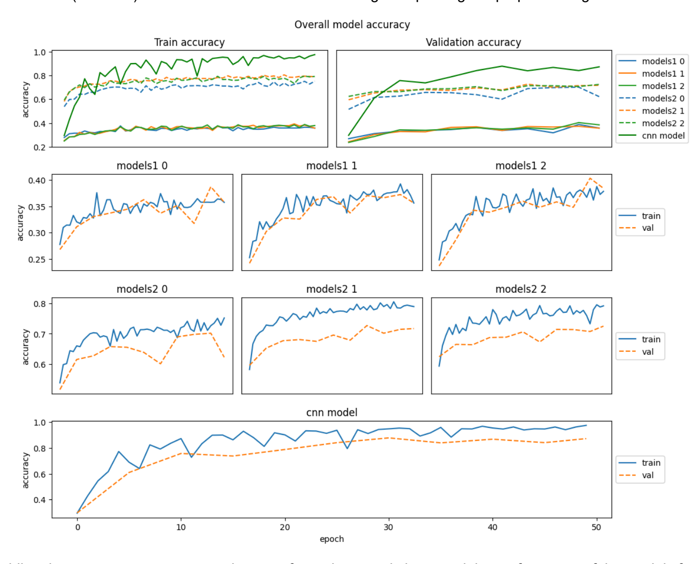
Figure 12: 3 Spectrogram sample example

As the training time for the SGD classifiers was relatively high, I decided to add these samples only in the CNN model as it was computed using a GPU.
the architecture of the CNN model was as follows

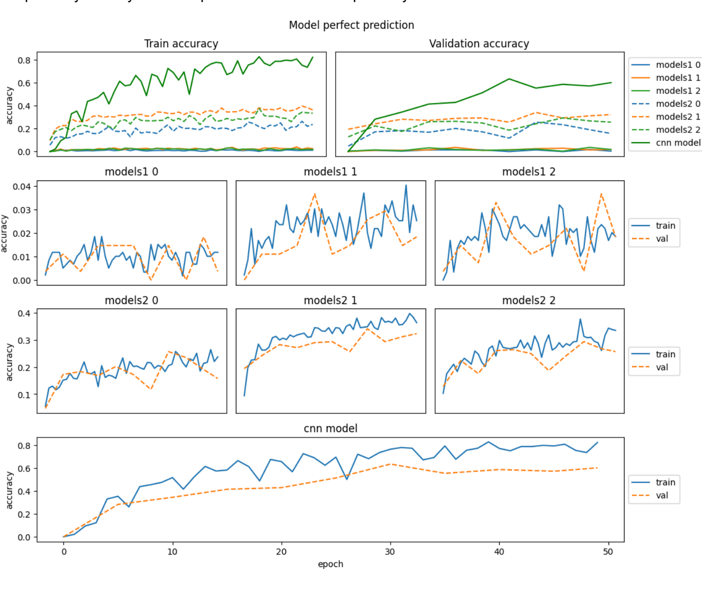
input shape	Composer	Instrument
(4,128,312)	Conv (in=4,out=16,k=(3,7),stride=(1,2)) ReLu BatchNorm	Conv (in=4,out=16,k=(3,7),stride=(1,2)) ReLu BatchNorm
(16,128,144)	Conv (in=16,out=16,k=(3,5),stride=2) ReLu BatchNorm	Conv (in=16,out=16,k=(3,5),stride=2) ReLu BatchNorm
(16,64,64)	Conv(in=16,out=32,k=3) ReLu MaxPool BatchNorm	Conv(in=16,out=32,k=3) ReLu MaxPool BatchNorm
	Conv(in=32,out=32,k=3) ReLu BatchNorm	
(32,32,32)	Conv(in=32,out=64,k=3) ReLu MaxPool BatchNorm	Conv(in=32,out=64,k=3) ReLu MaxPool BatchNorm
	Conv(in=64,out=64,k=3) ReLu BatchNorm	
(64,16,16)	Conv(in=64,out=128,k=3) ReLu MaxPool BatchNorm	Conv(in=64,out=128,k=3) ReLu MaxPool BatchNorm
	Conv(in=128,out=128,kernel=(3,3)) ReLu BatchNorm	
(128,8,8)	FullyConnected(8192,7)	FullyConnected(8192,10)

Model comparison

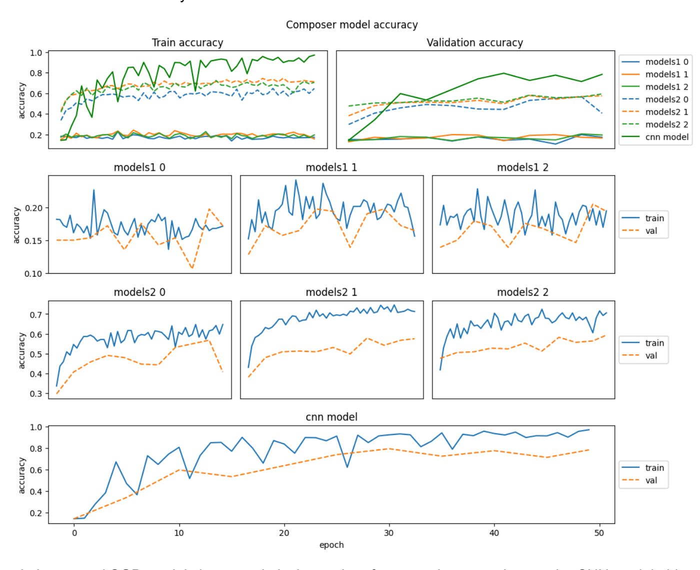
The overall accuracy of every model is defined by the combined accuracy of classifying both the correct composer and which instruments were playing, the first models (models1) are the SGD classifiers before adding the preprocessing step, the second models (models2) are the SGD classifiers after adding the spectrogram preprocessing



Clearly, adding the spectrogram preprocessing transformation greatly improved the performance of the models for both classification and validation. Next we have a perfect classification accuracy, that is the amount of samples a model managed to perfectly classify both composer and instruments perfectly at the same time.



A perfect prediction is much harder to get as the model has to perfectly classify 10 instruments (playing or not playing) as multiple instruments could be playing in the same audio file, and also correctly classify the composer. This metric is better suited for our model as we want a correct multioutput, not a singular prediction. Using this metric we can clearly see that the CNN has a much better accuracy over the second SGD models for each individual class.



As expected, the second SGD models have a relatively good performance in comparison to the CNN model, this was expected as shown previously, the SGD model has performance issues in the multioutput case which we can see here

Figure 13: Instrument classification accuracy for each class - SGD

The SGD models have a very erratic and rigid learning curve, it seems like they are improving with every epoch, but after 50 epochs, they still didn't manage to get any outstanding accuracies except for instrument 1 and 3 that have higher accuracies then the rest. We mostly care about a perfect accuracy when it comes to instruments, therefore the performance isn't too great despite having a high accuracy score on most instruments.

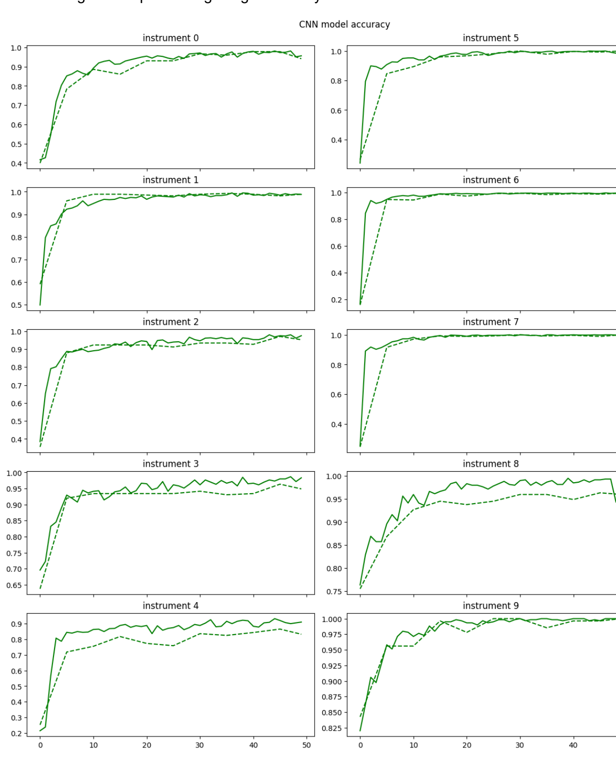
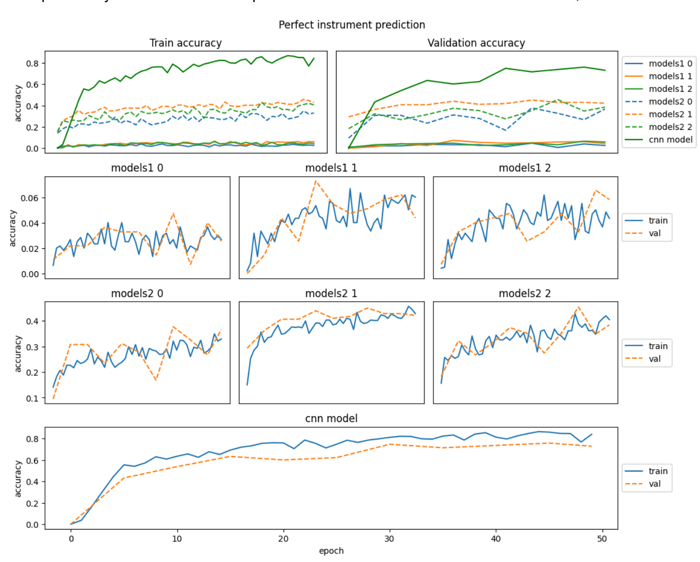
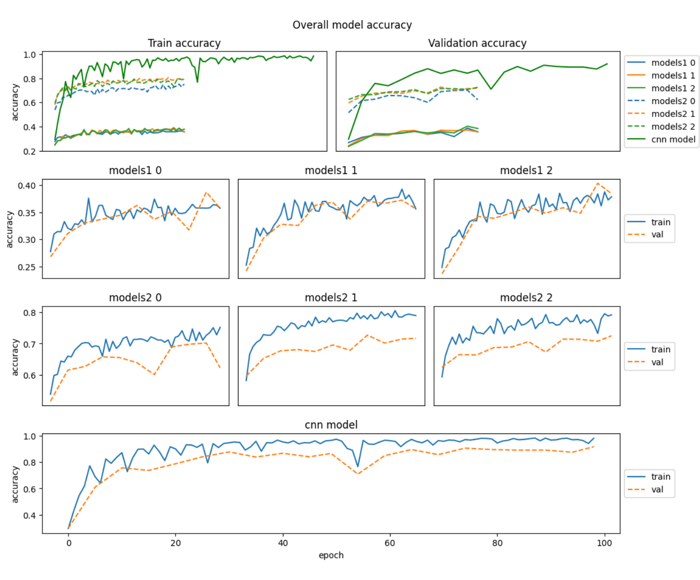


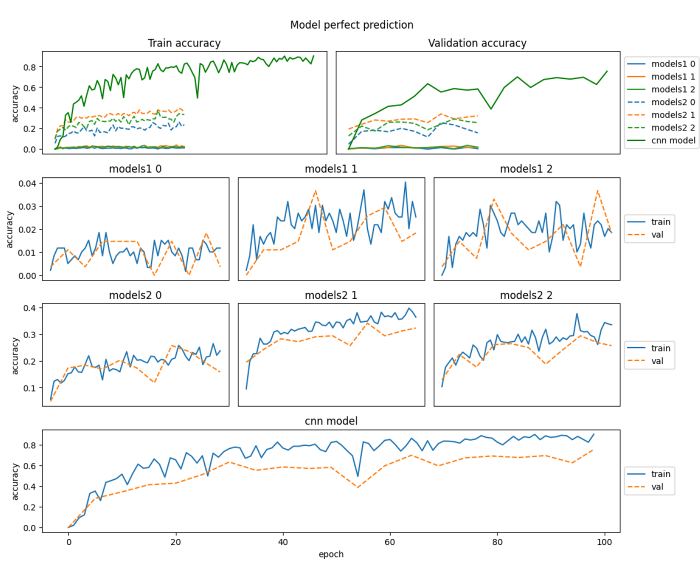
Figure 14: Instrument classification accuracy for each class - CNN

The CNN model manages to learn in a much more stable way and has better accuracies overall in comparison to the SGD models, this explains why it had a much better perfect classification score than the SGD models, we can see so here



As the CNN model is performing much better in the perfect prediction case, it was chosen to evaluate against the test data, but first it was trained on an additional 50 epoches





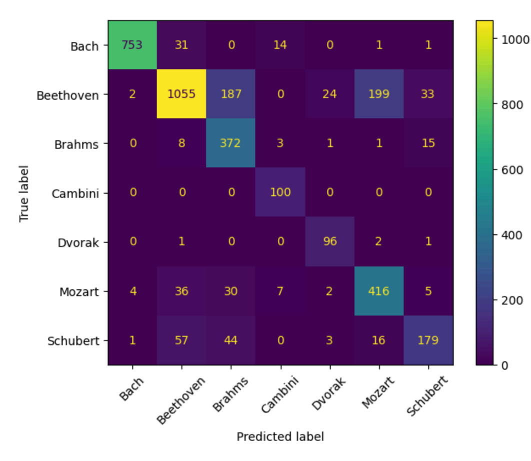
The CNN model seemed to have a large dip in performance at around epoch 53, and in general doesn't have a stable learning curve, this might indicate on a learning rate that is too high.

Test results

Since the dataset contained a very small number of test data, I resampled 5 seconds samples 100 times from each test sample, this was done to get a more precise real world accuracy, as the classification is done by only a 5 second sample from a longer audio file.

Composer classification:

	precision	recall	f1-score	support
Bach	0.99	0.94	0.97	800
Beethoven	0.89	0.70	0.78	1500
Brahms	0.59	0.93	0.72	400
Cambini	0.81	1.00	0.89	100
Dvorak	0.76	0.96	0.85	100
Mozart	0.66	0.83	0.73	500
Schubert	0.76	0.60	0.67	300
accuracy			0.80	3700
macro avg	0.78	0.85	0.80	3700
weighted avg	0.83	0.80	0.81	3700

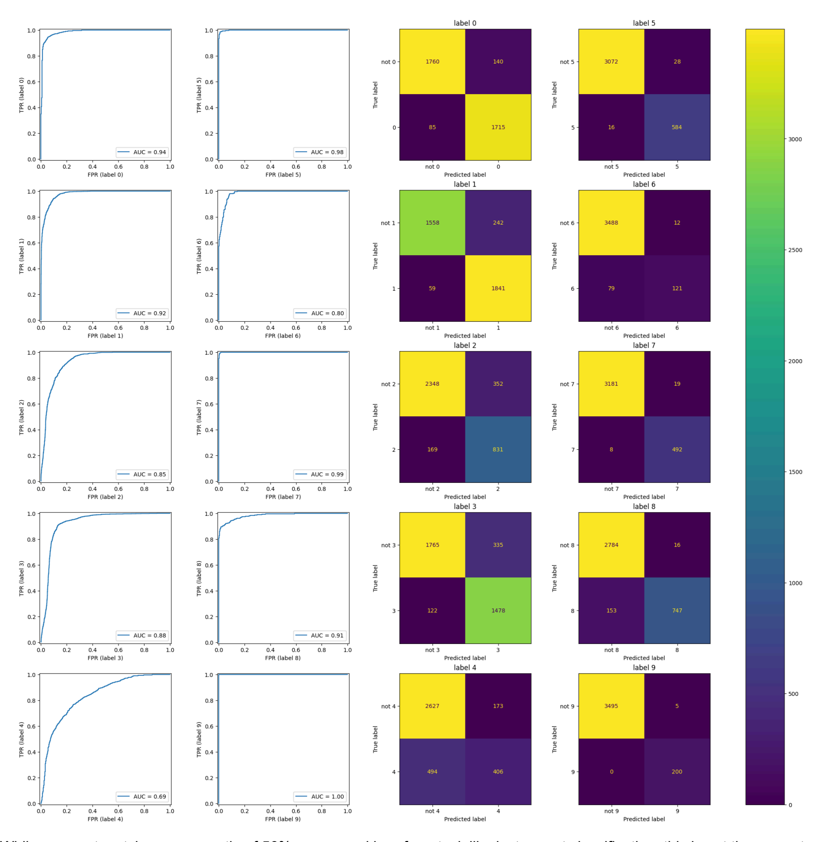


While the model had lower precision scores for Brahms and Mozart, it seems like they still had a high recall score, we can see that for the most part, audio files of pieces composed by Beethoven caused the misclassification, which just so happens to have the most samples overall. Additionally, the model underperforms in classifying Schubert as the composer. Overall the model performs very good especially in classifying the other 3 composers.
It is important to note that Beethoven had the most, and almost 3 times more unique samples then Bach, the composer with the second most sample in the training data, and for the most part, most of the errors the model made involved classifying Beethoven. Given the fact that the initial amount of training samples for all other composers were much lower, it is safe to say that the model performed very well.

Instrument classification:

	precision	recall	f1-score	support
0	0.92	0.95	0.94	1800
1	0.88	0.97	0.92	1900
2	0.70	0.83	0.76	1000
3	0.82	0.92	0.87	1600
4	0.70	0.45	0.55	900
5	0.95	0.97	0.96	600
6	0.91	0.60	0.73	200
7	0.96	0.98	0.97	500
8	0.98	0.83	0.90	900
9	0.98	1.00	0.99	200
micro avg	0.86	0.88	0.87	9600
macro avg	0.88	0.85	0.86	9600
weighted avg	0.86	0.88	0.86	9600
samples avg	0.90	0.90	0.88	9600

Exact Match Ratio: 0.5694594594594594



While an exact match accuracy ratio of 56% may sound low, for a task like instrument classification, this is not the case at all, and this stems from 2 main reasons. The first is that out of 10 classes, all 10 must be correct at the same time, even one misclassification ruins the entire result, a dummy classifier has an exact match accuracy score of only 0.5^{10}. The second reason, is that many instruments sound similar to other instruments, When loading the data, the classes for instruments were $[0, 40, 41, 42, 45, 60, 68, 70, 71, 73]$, these numbers refer to a specific instrument, and to a type of instrument. And it just so happens that the classes for labels 1,2,3 and 4 are Violin (40), Viola (41), Cello (42), and Pizzicato (45), all of them are strings! By grouping up only the labels for strings we get

	precision	recall	f1-score	support
0	0.92	0.95	0.94	1800
1	0.98	0.96	0.97	2400
2	0.98	0.96	0.97	2400
3	0.98	0.96	0.97	2400
4	0.98	0.96	0.97	2400
5	0.95	0.97	0.96	600
6	0.91	0.60	0.73	200
7	0.96	0.98	0.97	500
8	0.98	0.83	0.90	900
9	0.98	1.00	0.99	200
micro avg	0.97	0.94	0.96	13800
macro avg	0.96	0.92	0.94	13800
weighted avg	0.97	0.94	0.96	13800
samples avg	0.98	0.96	0.96	13800

Exact Match Ratio: 0.8472972972972973

Most errors seem to come from confusing one type of string instrument for another.

Model overall performance

	precision	recall	f1-score	support
Bach	0.99	0.94	0.97	800
Beethoven	0.89	0.70	0.78	1500
Brahms	0.59	0.93	0.72	400
Cambini	0.81	1.00	0.89	100
Dvorak	0.76	0.96	0.85	100
Mozart	0.66	0.83	0.73	500
Schubert	0.76	0.60	0.67	300
0	0.92	0.95	0.94	1800
1	0.88	0.97	0.92	1900
2	0.70	0.83	0.76	1000
3	0.82	0.92	0.87	1600
4	0.70	0.45	0.55	900
5	0.95	0.97	0.96	600
6	0.91	0.60	0.73	200
7	0.96	0.98	0.97	500
8	0.98	0.83	0.90	900
9	0.98	1.00	0.99	200
micro avg	0.85	0.86	0.85	13300
macro avg	0.84	0.85	0.84	13300
weighted avg	0.85	0.86	0.85	13300
samples avg	0.87	0.88	0.86	13300

Exact Match Ratio: 0.5108108108108108

In conclusion, despite being trained on a very small number of unique sample data, the final model still manages to get a very impressive Exact match ratio of 51% and an unweighted 85% overall accuracy in predicting both composer and instruments being played in an audio file.