KTH ROYAL INSTITUTE
OF TECHNOLOGY

Degree Project in Computer Science and Engineering

First cycle, 15 credits

Comparison of CNN and LSTM for classifying short musical samples

VICTOR STENMARK
TORE NYLÉN

Stockholm, Sweden, 2023

Comparison of CNN and LSTM for classifying short musical samples

VICTOR STENMARK
TORE NYLÉN

Degree Project in Computer Science and Engineering, First Cycle
15 credits
Date: January 30, 2024
Supervisor: Erik Fransén
Examiner: Pawel Herman
Swedish title: Jämförelse av CNN och LSTM för instrumentklassificering
School of Electrical Engineering and Computer Science

Abstract

Applying machine learning to music and audio data is becoming increasingly common. One such area of research is instrument classification, which is the task of identifying the instrument played in a given audio file. In this study, we compared two machine learning model types, LSTM and CNN, on the task of classifying ten different instruments. Additionally, hyperparameter tuning was performed to optimize three parameters used in each model. To test the performance of the models, two test datasets containing instrument samples were used. These were obtained from the NSynth dataset. The first was a randomized equally split dataset based on the NSynth training data. The other one was a curated test dataset created by the NSynth authors. The results for the optimized models showed that the CNN's accuracy on the randomized dataset was 97.1% and for the LSTM it was 96.0%. The corresponding results for the NSynth test dataset were 74.3% and 71.6%. While the results indicate that the CNN is slightly better, the difference could also be explained by the selected feature extraction technique or more optimized hyperparameters for the CNN. Furthermore, while the accuracy was higher for the CNN on both datasets, a Wilcoxon signed rank test showed that the difference was not statistically significant.

Sammanfattning

Att använda maskininlärning för musik- och ljuddata blir allt vanligare. Ett exempel på ett sådant forskningsområde är instrumentklassificering. Instrumentklassificering innebär att identifiera vilket instrument som spelas i en ljudfil. I denna studie jämfördes två typer av maskininlärningsmodeller, LSTM och CNN, givet denna uppgift. Därtill utfördes hyperparameteranpassning för att optimera tre parametrar i vardera modell. För att testa modellernas prestanda användes två datamängder tagna från NSynth-datamängden. Den ena var en slumpad datamängd byggd från NSynth-datamängdens träningsdata, medan den andra var en testdatamängd skapad av NSynth-författarna. Resultatet från de optimerade modellerna visade att CNN:en hade en precision på 97.1% på den slumpade datamängden och 74.3% på NSynth-testdatamängden. Motsvarande resultat för LSTM:en var 96.0% och 71.6%. Medan resultaten indikerar att CNN:en presterar bättre kan skillnaden möjligtvis förklaras av den valda metoden för att bearbeta ljuddatat. Alternativt kan det bero på mer optimerade hyperparametrar i CNN-modellen. Dessutom bör det poängteras att CNN:ens högre precision inte är statistiskt säkerställd, även om CNN:en var bättre på båda datamängderna.

Contents

Chapter 1

Introduction

As internet usage has increased, so has the number of songs on music platforms online. Websites such as YouTube and SoundCloud allow users to upload large amounts of content easily. This paired with the widespread use of music programs has made the amount of music data available online grow rapidly. Consequently, classifying and indexing all these songs have become increasingly difficult without the help of automated systems.

One approach to the problem of automatic music classification has been utilizing machine learning. This is because of the adaptability and complex pattern recognition that machine learning models possess. A use case of this includes Facebook's Demucs[1][2], a music source separation program that can separate a song into its original components.

A principal problem in the field of music classification is instrument classification. This is the problem of determining the instrument played in an audio file. Two common approaches to this and similar problems in music classification involve using Long Short-Term Memory (LSTM) models and Convolutional Neural Networks (CNN). This study aims to compare these two machine learning approaches given the task of classifying instrument-based audio samples.

1.1 Research Question

Which of the two machine learning approaches, CNN and LSTM, performs better for the task of musical instrument classification?

Chapter 2

Background

2.1 Supervised Learning

Supervised learning is a machine learning technique that makes use of pre-labeled data to make predictions. During training the algorithm trains on input data and the corresponding correct labels. The algorithm works by predicting the label based on the input data and the feedback from this process changes the parameters in the network. Then during validation and testing, the network predicts the labels of data that it has not been trained on, known as validation and test data. This provides an insight into the network's overall accuracy [3].

One iteration of the training process is known as an epoch and training is usually done over several epochs. During training, it is the model's learning rate that affects how much the model adapts to the data every epoch. The learning rate is a configurable value usually set between 0 and 1 [4]. A high learning rate allows the model to quickly adapt to the data, which can reduce the number of necessary epochs. However, a learning rate set too high can make the model overfit the training data. This means that the model over-adapts to patterns in the training data, making the model less performant on unseen data [5].

When training the model, the optimal configuration is often not known beforehand. The model's configuration consists of several manually chosen values that affect the learning process called hyperparameters. To improve the model, hyperparameter tuning can be used, which is the process of finding more optimal hyperparameter values. It works by executing the training process multiple times while adjusting the values of the hyperparameters [5].

2.2 Convolutional Neural Network (CNN)

A Convolutional Neural Network (CNN) is a model architecture commonly used in supervised learning, primarily for the task of image classification. CNNs typically make use of four different types of layers: convolutional layers, pooling layers, fully-connected dense layers, and dropout layers. The convolutional layer is the primary method of feature extraction in CNNs and aims to extract different features from the input sample [6]. The pooling layer is used to increase the computational performance of the network by downsampling the data by a factor. This decreases the number of parameters in the network which reduces the computational complexity [7]. A fully-connected layer is one where all neurons are connected to every neuron in the subsequent layer [8]. Lastly, there are dropout layers that randomly drop a certain amount of nodes during training, which helps reduce overfitting [9].

2.3 Long Short-Term Memory (LSTM)

A Long Short-Term Memory (LSTM) network is a type of Recurrent Neural Network (RNN) that allows for cycles in its node structure. This means that it allows for feedback to previous nodes for later iterations through the network. Additionally, nodes in an LSTM layer have states that act as a memory that can be altered. This makes it suitable for recognizing patterns with changes over time in sequences of data, such as audio. Similar to a CNN, an LSTM can also contain different types of layers such as dropout and dense layers [10].

2.4 Mel-spectrogram & MFCC

When training a model on audio there are multiple approaches for feature extraction. One such approach is to make a spectrogram, which is a visual display of the frequencies that make up the audio signal. They are created by using multiple Fourier transforms that are computed in a sequence of time intervals of the entire audio signal [11]. A mel-spectrogram is a spectrogram that has been weighted in its visual bands to better display relative differences in accordance with human hearing [12].

A Mel-frequency cepstrum (MFC) is the result of performing additional Fourier-related transforms on a mel-spectrogram. This in turn finds patterns in it. The distinct values from this process visualize the harmonic content of the

sound. Using these values one can create Mel-frequency cepstrum coefficients (MFCC) that compact the information even further [12] [13].

2.5 Related work

There has been a lot of previous work in the field of instrument classification. Kawwa [14] used various machine learning techniques including Naive Bayes, Random Forests, Support Vector Machines, and CNN. These were used together with the NSynth dataset, a dataset containing thousands of unique instrument samples. As preprocessing they used, among other methods, mel-spectrograms, chroma energy, spectral contrast, and MFCCs. For the CNN however, Kawwa only used mel-spectrograms to reduce training time and computing costs. With a set of ten instruments, Kawwa achieved the best accuracy with Random Forests at 65% followed by the CNN at 56%.

There are also examples of LSTMs used for instrument classification. Zhang and He [15] used an LSTM to classify clarinets, flutes, and trumpets. Their preprocessing involved transforming the samples into MFCCs. When applied to The University of Iowa Musical Instrument Samples, they achieved 85.6% accuracy with their LSTM.

Comparisons of CNNs and LSTMs in the audio domain have also previously been done. Gessle and Åkesson [16] compared a CNN with an LSTM in identifying the genres of songs. The CNN consisted of five convolutional layers followed by a dense layer and an output layer. Both the convolutional layers and the dense layer were followed by dropout layers. The LSTM was constructed with five LSTM layers, with a dropout layer after each. The models were tested on the GTZAN and FMA datasets, two datasets containing songs categorized into different genres. When trained and tested on data from ten different genres, their CNN model had an accuracy of 56.0% on the GTZAN dataset and 50.5% on the FMA dataset. Their LSTM model performed slightly worse with an accuracy of 42.0% and 33.5% respectively.

Chapter 3

Methods

3.1 Dataset

The NSynth dataset created by Engel et al. [17] has 305,979 solo recordings from 1,006 instruments gathered from commercial sample libraries. Each sample is four seconds long, monophonic, and has a sample rate of 16kHz. Both acoustic and synthetic samples are included in the dataset. The dataset contains samples of 11 different instrument classes split into three sub-datasets: training, validation, and testing. The training dataset contains 289,205 samples that do not appear in the test or validation datasets. The validation dataset contains 12,678 samples and the test dataset, which partially is a subset of the validation dataset, contains 4,096 samples. Neither of the NSynth sub-datasets is balanced in regard to the number of samples in each instrument category. A large number of samples across multiple instrument categories makes the NSynth dataset suitable for this study.

3.2 Implementation

3.2.1 Feature extraction

Feature extraction was made to optimize the data for the models. Every instrument category in the NSynth dataset except synth lead was included in the study. The synth lead category was excluded due to the category not being present in the NSynth test dataset. For each sample, the Librosa library [18] was used to extract the MFCCs. 13 coefficients at 126 timesteps of each

file were collected. MFCCs were chosen instead of mel-spectrograms in order to minimize memory usage during training. Furthermore, this allows for training on a larger amount of samples than what would be possible using mel-spectrograms.

3.2.2 5-fold cross validation

To reassure that the models' predictions are representative of the entire dataset, 5-fold cross-validation was done using the NSynth training dataset. Initially, 8,750 samples were extracted from each of the ten instrument categories. Afterward, the data was copied into five folds, each containing the same 87,500 samples. Each fold was split into training, validation, and test data with a split of 80/10/10%. The split was done so that the validation and test data were completely different in each fold. This results in the training data composition also being slightly different in every fold. The point of this is to test the models' ability to generalize. The models were then trained and evaluated on each of the five folds independently.

Since the same model architecture is trained on the five different folds independently, it will create five different models of the same architecture. These are distinct models but since they share the same architecture they will be grouped together to determine an average case.

3.2.3 Base models

Base CNN Model

The CNN model used by Kawwa [14] was implemented and its architecture was slightly adjusted. The last convolutional layer and subsequent max pooling layer were removed to reduce the number of parameters in the model. In table 3.1 the base CNN model architecture can be seen. It consists of two groups of two convolutional layers followed by max-pooling layers and dropout layers. The number of filters increases for each group. After the two groups, there is an additional convolutional layer and a dropout layer. In the end, there is a flatten layer to reduce dimensionality and lastly two dense layers.

Base LSTM Model

The base LSTM model, as seen in table 3.2, consists of five LSTM layers with dropout layers in between. Each LSTM layer has 32 units. After the final LSTM layer, there is a flatten layer to reduce dimensionality and a dense

Layer name	Layer type	Hyperparameters
conv2d	Conv2D	input_shape = (126, 13, 1), filters = 32, kernel_size = (3, 3), bias = 32, activation = ReLU
conv2d_1	Conv2D	input_shape = (126, 13, 32), filters = 32, kernel_size = (3, 3), bias = 32, activation = ReLU
max_pooling2d	MaxPooling2D	input_shape = (126, 13, 32), pool_size = (2, 2), strides = (2, 2)
dropout‡	Dropout	input_shape = (63, 6, 32), dropout_rate = 0.25
conv2d_2*	Conv2D	input_shape = (63, 6, 32), filters = 64, kernel_size = (3, 3), bias = 64, activation = ReLU
conv2d_3*	Conv2D	input_shape = (63, 6, 64), filters = 64, kernel_size = (3, 3), bias = 64, activation = ReLU
max_pooling2d_1	MaxPooling2D	input_shape = (63, 6, 64), pool_size = (2, 2), strides = (2, 2)
dropout_1‡	Dropout	input_shape = (31, 3, 64), dropout_rate = 0.25
conv2d_4†	Conv2D	input_shape = (31, 3, 64), filters = 128, kernel_size = (3, 3), bias = 128, activation = ReLU
dropout_2‡	Dropout	input_shape = (31, 3, 128), dropout_rate = 0.25
flatten	Flatten	input_shape = (31, 3, 128)
dense	Dense	input_shape = (11904), units = 512, bias = 512 activation = ReLU
dropout_3 ‡	Dropout	input_shape = (512), dropout_rate = 0.25
dense_1	Dense	input_shape (512), units = 10, bias = 10, output_shape = (10), activation = Softmax

Table 3.1: The base CNN model architecture where the layers are presented in order from top to bottom. The asterisks are representing the following hyperparameter variables: *: conv_2_filters_1, †: conv_2_filters_2, ‡: dropout_cnn.

layer with 512 nodes. Finally, it has a dropout layer and a dense layer with 10 nodes. The architecture of the LSTM was chosen to somewhat resemble the layer structure of the CNN.

Layer name	Layer type	Hyperparameters
lstm	LSTM	input_shape = (126, 13), units = 32, bias = 128, return_sequences = true, activation = TanH
dropout	Dropout	input_shape = (126, 32), dropout_rate = 0.5
lstm_1*	LSTM	input_shape = (126, 32), units = 32, bias = 128, return_sequences = true, activation = TanH
dropout_1‡	Dropout	input_shape = (126, 32) dropout_rate = 0.5
lstm_2*	LSTM	input_shape = (126, 32), units = 32, bias = 128, return_sequences = true, activation = TanH
dropout_2‡	Dropout	input_shape = (126, 32), dropout_rate = 0.5
lstm_3†	LSTM	input_shape = (126, 32), units = 32, bias = 128, return_sequences = true, activation = TanH
dropout_3‡	Dropout	input_shape = (126, 32), dropout_rate = 0.5
lstm_4†	LSTM	input_shape = (126, 32), units = 32, bias = 128, return_sequences = true, activation = TanH
flatten	Flatten	input_shape = (126, 32)
dense	Dense	input_shape = (4032), units = 512, bias = 512 activation = ReLU
dropout_4	Dropout	input_shape = (512), dropout_rate = 0.5
dense_1	Dense	input_shape = (512), units = 10, bias = 10, output_shape = (10), activation = Softmax

Table 3.2: The base LSTM model architecture where the layers are presented in order from top to bottom. The asterisks are representing the following hyperparameter variables: *: lstm_layer_units_1, †: lstm_layer_units_2, ‡: dropout_lstm.

3.2.4 Hyperparameters

For both the base CNN and the base LSTM, three hyperparameter variables were selected for hyperparameter tuning.

CNN hyperparameters

The CNN's hyperparameter variables consisted of the number of filters in the convolutional layers and dropout rates in the dropout layers. The variables were chosen due to previous research showing that they have a significant impact on performance [19][20]. The first variable, conv_2_filters_1, was the number of filters in layers conv2d_2 and conv2d_3 (as seen in table 3.1) which

varied between 32, 64, and 128 filters. The second variable, conv_2_filters_2, was the number of filters in the last convolutional layer, conv2d_4. It varied between 32, 64, 128, and, 256. Lastly, the dropout rate of every dropout layer was set by one variable, dropout_cnn. Its values were between the range of 0.1 and 0.5.

LSTM hyperparameters

The hyperparameters of the LSTM were the number of units in the LSTM layers and the dropout rates of the dropout layers. These were used because of previous research showing that they have a non-negligible impact on performance [21]. The variable lstm_layer_units_1 was the number of units in the lstm_1 and lstm_2 layers (as seen in table 3.2). Additionally, lstm_layer_units_2 was the number of units in the lstm_3 and lstm_4 layers. Both of these variables varied between 32, 64, and 128 units. Finally, the dropout rate of each dropout layer excluding the last one was determined by dropout_lstm. It was set to values in the range of 0.1 and 0.5.

3.2.5 Training setup

The training of the CNN and the LSTM was done five times each, one time for each fold. Each fold generated one sub-model. For both the CNN and the LSTM, these five sub-models were then averaged to get a generalized result.

The training for each fold of the CNN was done over 50 epochs using sparse categorical cross-entropy as the loss function. The learning rate was set to 0.0001 and the batch size to 32. The folds in the LSTM model had identical parameters except for the learning rate which was set to 0.001. During training, the models are evaluated on the validation data after each epoch. Since the models are not trained on this data, this provides an insight into the models' ability to generalize.

3.2.6 Hyperparameter tuning

The RandomSearch tuner in Keras [22] was used to optimize the hyperparameters in both the CNN and the LSTM. The objective set for the tuner was to maximize the accuracy on the validation data. To speed up the tuning, only the first fold of the 5-fold dataset was used in the hyperparameter search. Five combinations of the three hyperparameter values were tested with two executions each. Every execution consisted of training the model with the selected

hyperparameters. The average of the two executions is what then forms the score. The small number of trials and executions was chosen to limit the already substantial training time.

Having run the hyperparameter search, the hyperparameters in the models with the highest scores were used in new optimized models. The training of these optimized models was done using the same conditions as their base counterparts, as described in section 3.2.5.

3.3 Evaulation and testing

Both the 5-fold test dataset and the NSynth test dataset were used for testing. A number of different criteria were used to determine the models' performance. These include loss, accuracy, as well as a confusion matrix. The average over the five folds for each of these metrics is calculated. Wilcoxon signed-rank tests were then performed on pairs of the models on the 5-fold test dataset. This was done to display the significance level of the differences between the models.

Chapter 4

Results

For the hyperparameter search the results are presented in two tables showing the hyperparameter values and their corresponding accuracy score. Following that there are graphs showing the accuracy and loss from training as well as confusion matrices from predictions on the 5-fold test dataset and the NSynth test dataset. The results from the optimized versions of the CNN and the LSTM are shown.

It should be noted that all results in section 4.2 are averages of the results from the five sub-models of the LSTM and CNN. Every sub-model is coupled to a specific fold in the 5-fold dataset.

4.1 Hyperparameter results

As seen in table 4.1 and 4.2, the hyperparameters have a non-trivial impact on the score which is the model's validation accuracy. Lower dropout and a larger number of filters appear to improve the overall score of the CNN. For the LSTM, the number of nodes and the dropout do not seem to impact the score as much. Trial 5 had the highest score among the CNN configurations, which meant that its values were used in the optimized CNN model. Similarly, the values of trial 4 were used for the optimized LSTM model since it had the best score in its hyperparameter tuning.

Hyperparameters:	**Trial 1**	**Trial 2**	**Trial 3**	**Trial 4**	**Trial 5**
dropout_cnn	0.3	0.5	0.4	0.2	0.2
conv_2_filters_1	32	64	64	128	128
conv_2_filters_2	256	32	128	256	128
Score:	0.9584	0.9127	0.9616	0.9782	0.9802

Table 4.1: The results of the hyperparameter tuning on the CNN model

Hyperparameters:	**Trial 1**	**Trial 2**	**Trial 3**	**Trial 4**	**Trial 5**
dropout_lstm	0.4	0.2	0.3	0.3	0.4
lstm_layer_units_1	128	32	64	32	64
lstm_layer_units_2	64	32	32	128	128
Score:	0.9588	0.9584	0.9567	0.9657	0.9607

Table 4.2: The results of the hyperparameter tuning on the LSTM model

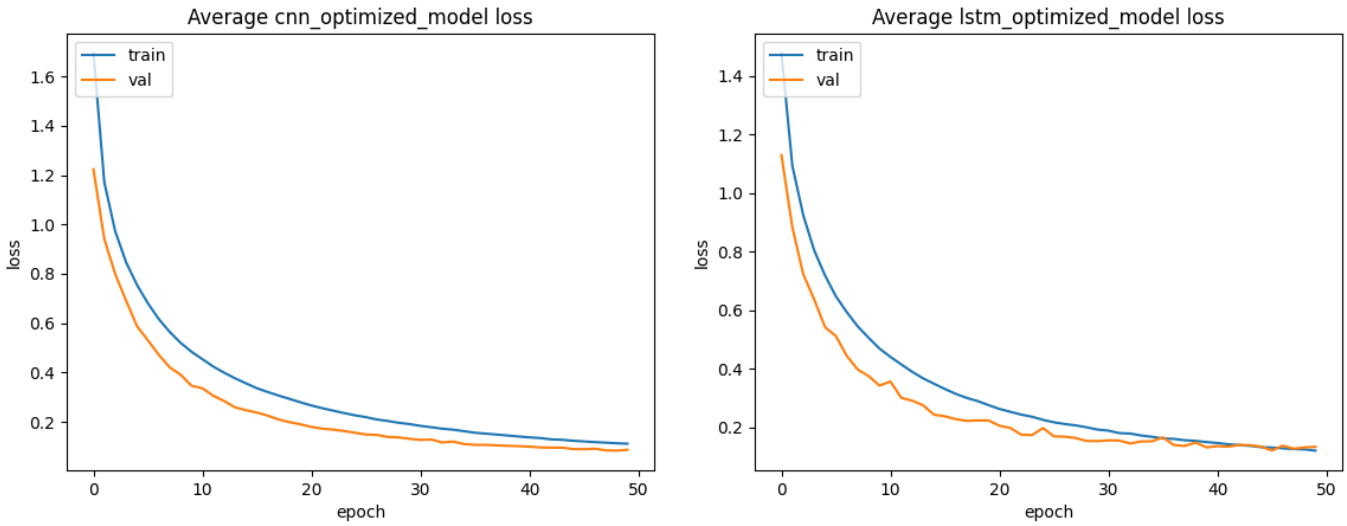
4.2 Model results

Model Name:	**5-fold loss**	**5-fold accuracy**	**NSynth loss**	**NSynth accuracy**
Base model CNN	0.112	0.963	2.220	0.732
Optimized CNN model	0.090	0.971	2.465	0.743
Base model LSTM	0.201	0.932	2.138	0.693
Optimized LSTM model	0.136	0.960	2.372	0.716

Table 4.3: The loss and accuracy of each model averaged over the 5 folds

Table 4.3 consists of the losses and accuracies from testing the trained models on the 5-fold test dataset and NSynth test dataset. The values show the averages of the scores over the five folds of each model. It shows that the hypertuned models had higher overall accuracy and lower loss than their base model counterparts. On the NSynth dataset, all models had substantially higher losses than the 5-fold dataset, as well as lower accuracies.

4.2.1 Loss and accuracy

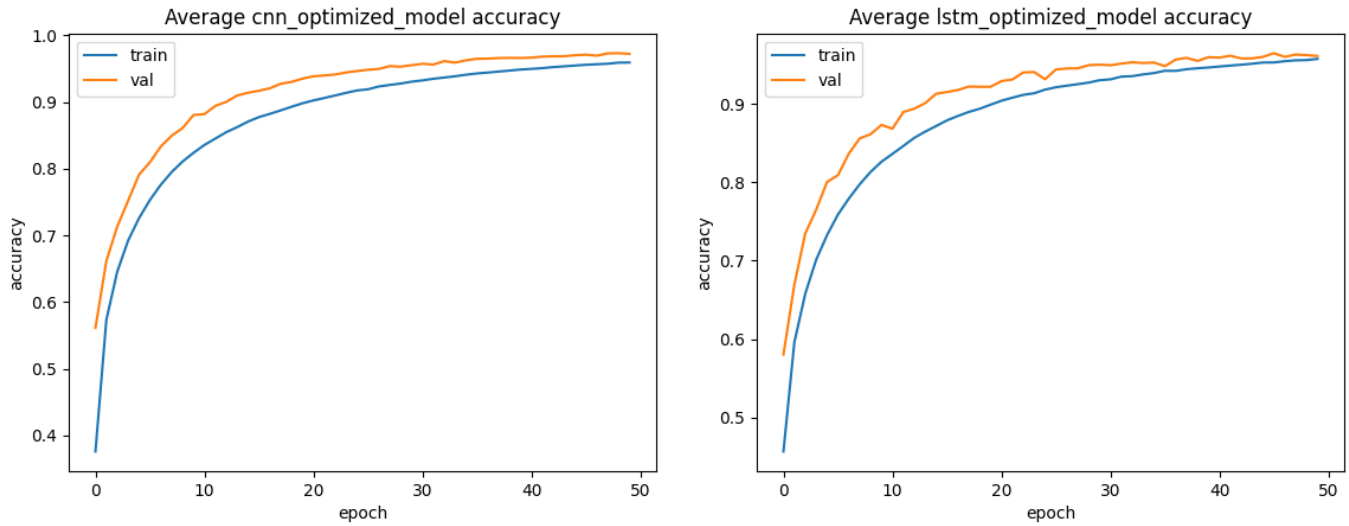


(a) Averaged loss of the optimized CNN model for the 5-fold dataset

(b) Averaged loss of the optimized LSTM model for the 5-fold test dataset

Figure 4.1: Averaged loss history for the optimized models

Figures 4.1a and 4.1b show the optimized models' loss for both the training and validation data over each epoch. The values are averaged over the five folds. The training and validation loss of both models consistently decreases until it stagnates around epoch 40 (Fig. 4.1a). The validation loss is lower than the training loss for both models over most epochs. This is most likely due to the dropout layers in each model.



(a) Averaged accuracy of the optimized CNN model for the 5-fold dataset

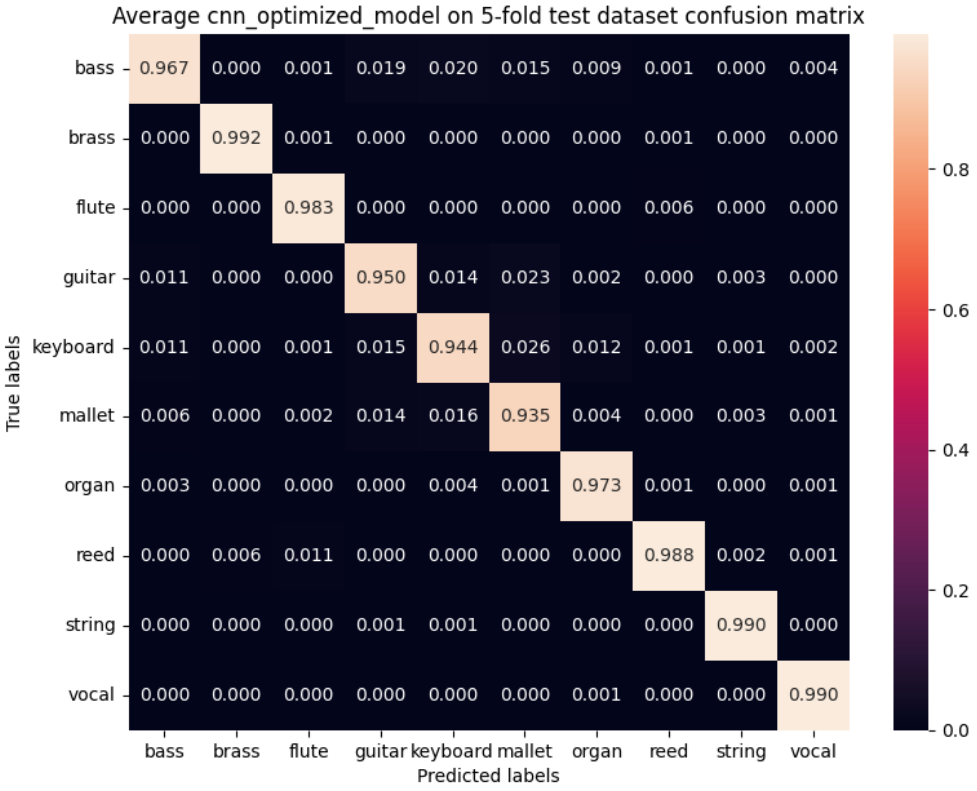
(b) Averaged accuracy of the optimized LSTM model for the 5-fold test dataset

Figure 4.2: Averaged accuracy for the optimized models

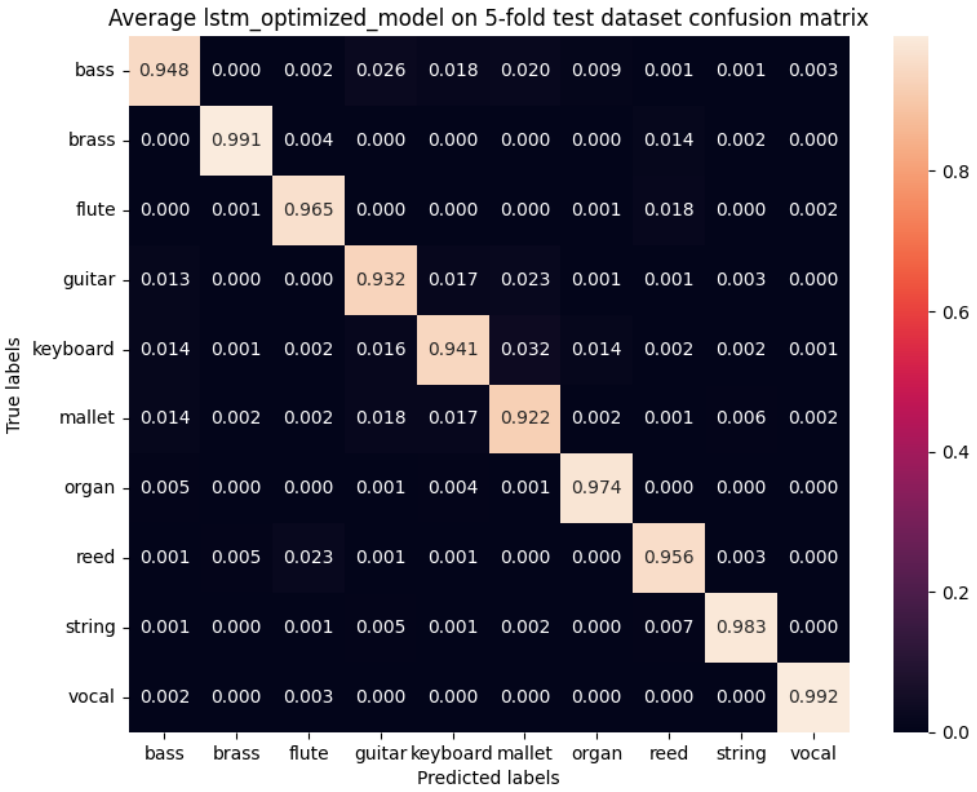
The graphs in figure 4.2 show the accuracy of each model after every training epoch. Similar to the loss graphs in Fig. 4.1, the curves closely resemble each other. In both models, the validation accuracy is consistently higher than the training accuracy. Again, this is most likely due to the dropout layers. Additionally, both models' training accuracy and validation accuracy increases in tandem which indicates that neither model is overfitting.

4.2.2 Confusion matrices

Each confusion matrix in figure 4.3 and 4.4 consists of a grid with the true labels on the y-axis and the predicted labels on the x-axis. The number in the cell (x_i, y_j) represents the proportion of all x_i-predictions that have the true label of y_j. This means that the true positives can be found on the left-to-right descending diagonal. The color scheme is designed so that a higher number is colored with a lighter color and a lower number with a darker color.

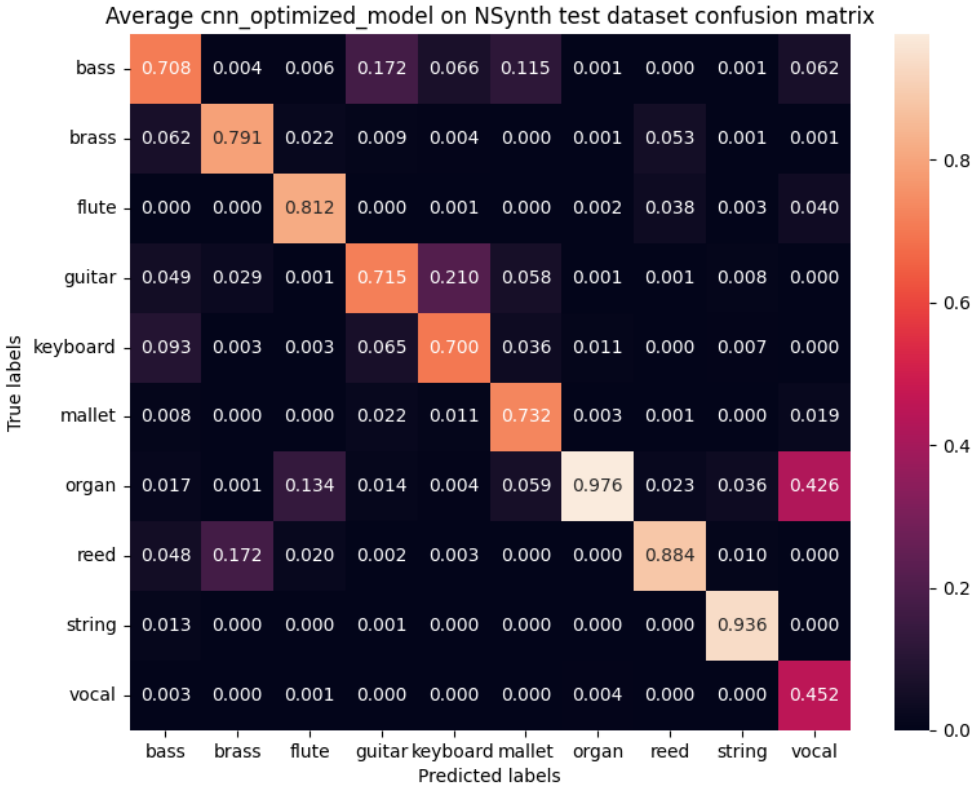


(a) Averaged confusion matrix for the optimized CNN model on the 5-fold test dataset

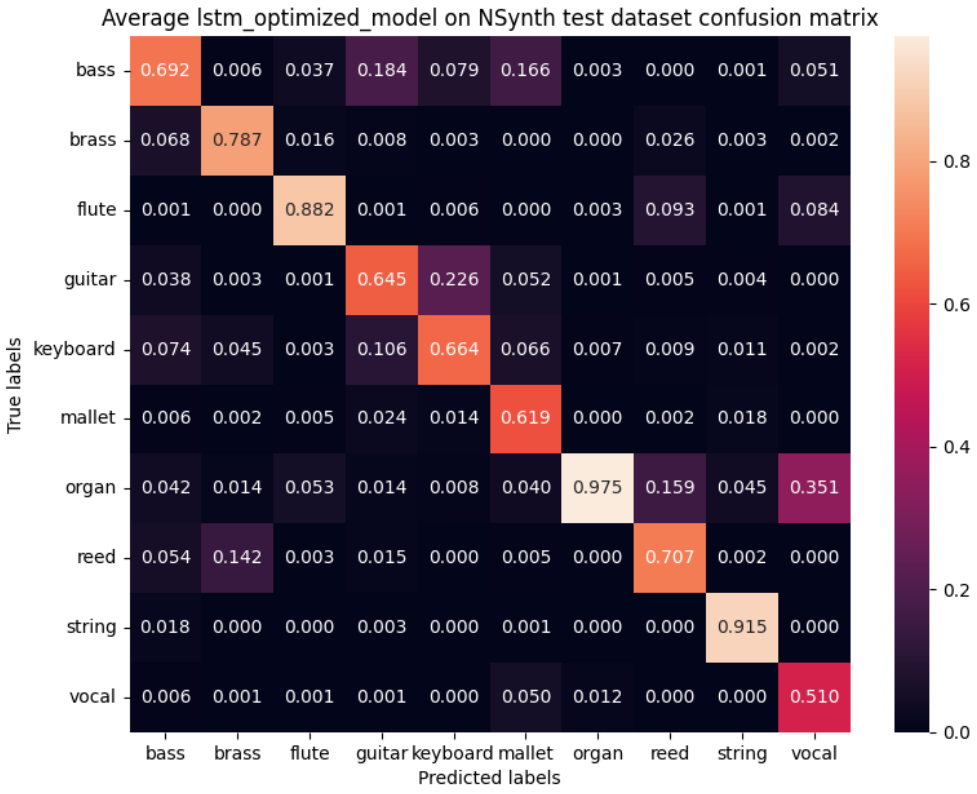


(b) Averaged confusion matrix for the optimized LSTM model on the 5-fold test dataset

Figure 4.3: Averaged confusion matrices on the 5-fold test dataset for both optimized models



(a) Averaged confusion matrix for the optimized CNN model on the NSynth dataset



(b) Averaged confusion matrix for the optimized LSTM model on the NSynth dataset

Figure 4.4: Averaged confusion matrices on the NSynth test dataset for both optimized models

As can be seen in figure 4.3, the models performed evenly around 90-99% across all instrument categories on the 5-fold dataset. The LSTM, however, had slightly lower accuracy than the CNN across all instrument classes except vocal and organ. Furthermore, the differences between the models for these two classes were minor at 0.2% and 0.1%.

On the NSynth dataset, however, the models' accuracies were a lot lower with a range between 45% and 98%, as seen in figure 4.4. Most instrument classes were in the 60-80% range for the LSTM and 70-80% for the CNN. Two outliers with high performance for both models were organ, and string, with accuracies of around 98% and 93%. Both models struggled with classifying vocal samples, mistaking them for organs. Only 51% of the vocal samples were correctly identified by the LSTM, and the performance was even worse with the CNN at 45%. Another difference between the models' performance was in the reed category, where the CNN had 18 percentage points better accuracy than the LSTM.

4.3 Wilcoxon signed-rank test

First model	Second model	p-value
Base CNN Model	Optimized CNN Model	0.0625
Base LSTM Model	Optimized LSTM Model	0.0625
Optimized LSTM Model	Optimized CNN Model	0.0625

Table 4.4: Wilcoxon signed-rank test of three combinations of the models

Table 4.4 shows identical values for the Wilcoxon signed-rank tests on three combinations of models. Typically if the p-value is less than 0.05, the models can be considered to be significantly different [23]. With a p-value of 0.0625 for all comparisons, this means that it cannot be statistically shown that one model is better than the other.

For every comparison in table 4.4, the second model had higher performance on all 5 folds. This means that the rank assigned to the folds will be consistently positive and that the sum of the negative ranks will always be 0. Consequently, this results in the Wilcoxon signed-rank tests being equal for all comparisons in table 4.4.

Chapter 5

Discussion

5.1 Input representation

The MFCC was chosen as the input representation due to its compactness compared to other representations. MFCCs with a limited number of coefficients are particularly suited for harmonic instruments, such as string instruments and voices [12]. Certain instruments, such as mallets, have a more inharmonic overtone structure, meaning that MFCCs might not encode that information as well. For these instances, a mel-spectrogram would perhaps be better suited. Furthermore, a mel-spectrogram contains more information overall, which could result in finer details being recognized by the classifiers.

5.2 LSTM and CNN

While not statistically shown that the optimized CNN is better than the optimized LSTM model, it did perform better on both datasets over all five folds. The same is true in the comparison between the base CNN and LSTM models. There are a few possibilities as to why this might be the case. One is the structure of the samples and the selected feature extraction technique. Since all samples are four seconds long it is appropriate as an image for the CNN to process it efficiently. If the samples were longer and contained more data, that might not necessarily still be the case.

It could also be the case that we found more optimized hyperparameters for the CNN than for the LSTM, meaning that if the LSTM was better tuned it could have reached similar results. The same is true for the overall architecture of the

models, especially since we used a provided CNN model that had previously been tested while the LSTM was made from scratch.

5.3 Differences in samples and instrument categories

What's noteworthy about the results regarding instrument categories is that a few categories had really high accuracies across all models and datasets. These include the organ and the string categories. This could perhaps be due to the distinct sonic qualities of those instrument categories. Both of these instrument categories allow for tones that are sustained with a consistent harmonic structure, while instruments such as the bass and mallets have a rapid decay in their spectrum. However, the difference could also be explained by a larger number of easy samples of those categories in both datasets.

5.4 Hyperparameter tuning

Due to the limited number of hyperparameters tested, it is difficult to draw any definite conclusions about their impact on performance. Only trying five different combinations of hyperparameters is not nearly enough to ensure that the optimal values have been found. Running only two executions for each combination on one hyperparameter configuration with a high dropout means the randomness involved could change the perceived performance. Therefore a high accuracy in the hyperparameter tuning might not necessarily result in equally high testing accuracy. Similarly, running the hyperparameter tuning on only one fold could mean that the hyperparameters optimize only for that fold. Furthermore, there are many more hyperparameters that could be added which could have a non-negligible impact on the performance.

5.5 Performance gap between the test datasets

Both models had high performance when tested on the 5-fold test dataset and considerably lower performance on the NSynth test dataset. There are a number of possible explanations for this. One is that the NSynth test dataset is not equally distributed between the instrument classes. Furthermore, when listening to a number of wrongly classified samples, we had much difficulty correctly identifying the instrument class as well. Some of the misclassified

samples were synthetic which made them dissimilar to their acoustic counterparts.

5.6 Reliability of results

The entire dataset contained a lot of synthetic samples. In turn, this means that the model trains on these samples to recognize them as the instrument representation. This could mean that the model does not accurately predict real-life acoustic instruments as well. An improvement to the method could be to filter out the synthetic instruments if the goal is to classify acoustic instruments. Having a model that distinguishes between these categories as well could improve the overall accuracy.

The results from the Wilcoxon signed-rank test show that one model could not be statistically determined to be better than any other. This could be due to only having five folds in the k-fold cross-validation testing. Using more folds one could more accurately determine if there are any real performance differences between the models.

5.7 Previous research

When comparing the results to previous research, it seems as if our use of MFCCs instead of mel-spectrograms improved overall performance. Kawwa [14] used mel-spectrograms in conjunction with a CNN model similar to both of our CNNs. When tested on the NSynth test dataset Kawwa's CNN model received an accuracy of 56%. This is substantially lower than the results of both our base and optimized CNN models which had accuracies of 73% and 74%. Since the major difference was the feature extraction technique with MFCCs in our case and mel-spectrograms in Kawwa's case, it indicates that MFCCs are more suitable for instrument classification.

Previous research also suggests that CNNs have slightly higher accuracies than LSTMs in other fields of music classification. Gessle and Åkesson [16] compared CNNs and LSTMs on the problem of musical genre classification. Similarly to our study, their results showed higher accuracies on the CNN models for both of their datasets. This strengthens the notion that CNNs are more performant on musical classification problems compared to LSTMs.

5.8 Future work

As a baseline for future work, combining LSTM layers and convolution layers might prove even more performant than either of the separate models. Harnessing the CNNs ability of image recognition with the memory aspect of the LSTM could be interesting. Having the LSTM memory could also help when using longer samples, while the CNNs could potentially provide better accuracy. Another expansion could be, as mentioned above, to train the model to distinguish between acoustic and synthetic samples.

Chapter 6

Conclusions

In this report, we compared two machine learning techniques, LSTM and CNN, given the task of musical instrument classification. One model for each machine learning technique was created and hyperparameter tuning was performed on them. The data used in training and testing was extracted MFCCs from the NSynth dataset. Two separate datasets were used for evaluation, a randomized 5-fold test dataset and a curated NSynth test dataset.

When comparing the tuned CNN and LSTM models it was evident that their difference in performance was quite small. On the randomized 5-fold test dataset the average difference in accuracy was 1.1 percentage points in favor of the CNN. For the NSynth test dataset, the gap was slightly larger at 2.6 percentage points. However, as shown by the Wilcoxon signed rank test, the difference was not statistically significant. Nevertheless, the optimized CNN had higher accuracy than the LSTM across all five folds, suggesting that it could be shown to be statistically better with more folds.

This study shows that it is possible to achieve high accuracy using both CNNs and LSTMs for the problem of musical instrument classification. As to which approach is better, it cannot be conclusively determined by the result of this study.

Bibliography

[1] Simon Rouard, Francisco Massa, and Alexandre Défossez. "Hybrid Transformers for Music Source Separation". In: *ICASSP 23*. 2023.

[2] Alexandre Défossez. "Hybrid Spectrogram and Waveform Source Separation". In: *Proceedings of the ISMIR 2021 Workshop on Music Source Separation*. 2021.

[3] Erik G Learned-Miller. "Introduction to supervised learning". In: *I: Department of Computer Science, University of Massachusetts* (2014), p. 3.

[4] Jason Brownlee. *Understand the Impact of Learning Rate on Neural Network Performance*. 2019. URL: https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/.

[5] *Classification applications with deep learning and machine learning technologies*. eng. Studies in Computational Intelligence Ser. ; v.1071. Cham, Switzerland: Springer, 2023. ISBN: 9783031175763.

[6] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. "Understanding of a convolutional neural network". In: *2017 International Conference on Engineering and Technology (ICET)*. 2017, pp. 1–6. DOI: 10.1109/ICEngTechnol.2017.8308186.

[7] Jiuxiang Gu et al. "Recent advances in convolutional neural networks". In: *Pattern Recognition* 77 (2018), pp. 354–377. ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2017.10.013. URL: https://www.sciencedirect.com/science/article/pii/S0031320317304120.

[8] S.H. Shabbeer Basha et al. “Impact of fully connected layers on performance of convolutional neural networks for image classification”. In: *Neurocomputing* 378 (Feb. 2020), pp. 112–119. DOI: 10.1016/j.neucom.2019.10.008. URL: https://doi.org/10.1016/j.neucom.2019.10.008.

[9] Pierre Baldi and Peter J Sadowski. “Understanding Dropout”. In: *Advances in Neural Information Processing Systems*. Ed. by C.J. Burges et al. Vol. 26. Curran Associates, Inc., 2013. URL: https://proceedings.neurips.cc/paper_files/paper/2013/file/71f6278d140af599e06ad9bf1ba03cb0-Paper.pdf.

[10] Jose Mejia et al. “Prediction of time series using an analysis filter bank of LSTM units”. eng. In: *Computers & industrial engineering* 157 (2021), pp. 107371–. ISSN: 0360-8352.

[11] Monika Dörfler, Roswitha Bammer, and Thomas Grill. “Inside the spectrogram: Convolutional Neural Networks in audio processing”. In: *2017 International Conference on Sampling Theory and Applications (SampTA)*. 2017, pp. 152–155. DOI: 10.1109/SAMPTA.2017.8024472.

[12] Alexander Lerch. *An introduction to audio content analysis : applications in signal processing and music informatics*. eng. Hoboken, New Jersey: Wiley, 2012. ISBN: 9781118393550.

[13] Tom Bäckström et al. *Introduction to Speech Processing*. 2nd ed. 2022. DOI: 10.5281/zenodo.6821775. URL: https://speechprocessingbook.aalto.fi.

[14] Nadim Kawwa. *NSynth*. https://github.com/charlespwd/project-title. 2019.

[15] Chen Zhang and Ye He. *INSTRUMENT CLASSIFICAITON*. 2017. URL: https://hajim.rochester.edu/ece/sites/zduan/teaching/ece472/projects/2017/ChenZhang_YeHe_paper.pdf.

[16] Gabriel Gessle and Simon Åkesson. *A comparative analysis of CNN and LSTM for music genre classification*. 2019.

[17] Jesse Engel et al. *Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders*. 2017. eprint: arXiv:1704.01279.

[18] Brian McFee et al. *librosa/librosa: 0.10.0.post2*. Version 0.10.0.post2. Mar. 2023. DOI: 10.5281/zenodo.7746972. URL: https://doi.org/10.5281/zenodo.7746972.

[19] Abhinav Agrawal and Namita Mittal. “Using CNN for facial expression recognition: a study of the effects of kernel size and number of filters on accuracy”. eng. In: *The Visual computer* 36.2 (2020), pp. 405–412. ISSN: 0178-2789.

[20] Mikolaj Wojciuk et al. “The Role of Hyperparameter Optimization in Fine-Tuning of Cnn Models”. In: *SSRN Electronic Journal* (2022). DOI: 10.2139/ssrn.4087642. URL: https://doi.org/10.2139/ssrn.4087642.

[21] Nils Reimers and Iryna Gurevych. “Optimal Hyperparameters for Deep LSTM-Networks for Sequence Labeling Tasks”. In: *CoRR* abs/1707.06799 (2017). arXiv: 1707.06799. URL: http://arxiv.org/abs/1707.06799.

[22] François Chollet et al. *Keras*. https://keras.io. 2015.

[23] Maarten Grootendorst. *Validating your Machine Learning Model*. 2019. URL: https://www.maartengrootendorst.com/blog/validate/.