Separation of Drum and Bass from Monaural Tracks

Course	MSc in AI and Robotics
Module	Neural Networks (1022870)
Student Name	Jose Maria Salas
Student ID	1794530

1 Introduction

The aim of this project was to replicate the results from the provided research paper, in which a DRNN is evaluated as a configuration to separate a monaural source of bass and drums into two separate files.

The average training time of each Configuration is around 2 hours in an EC2 server with 8 cores and a GPU. This means that not all the desired configurations and ideas were tested.

2 Using The Code

2.1 Prerequisites

This code was developed in Python 3. The following libraries are required:

- Keras v2
- Tensorflow
- Sklearn
- Librosa
- mir_eval
- pydot
- graphviz
- prettytable
- h5py
- numpy
- matplotlib

2.2 Code files

There are a total of 4 python files:

- SCSS.py: main file
- AudioDataSet.py: class that generates the dataset.
- drnn.py: where all the neural network configurations are stored.
- plot_result.py: functions to plot the loss charts.

2.3 Dataset

The dataset can be placed anywhere; the default location is the folder where the Python files are.

The folder structure and naming convention should be as follows:

- Mixed audio: <dataset folder name>/seg_mix/
- Bass files: <dataset folder name>/seg_bass/
- Drums files: <dataset folder name>/seg_drums/
- Test files: <dataset folder name>/Test/

2.4 Experiment results

Results will be stored in a new folder under folder where the code is run. It will be in a format: “Experiment-<configuration number-time/”

2.5 Command Line

All arguments are optional

The following is the help text from the argument parser:

```
-h, --help                show this help message and exit
-dspath DSPATH            Dataset path [default: <code folder>/SCSS_DataSet/]
-exppath EXPPATH          Path where the experiment results is stored [default:
                          <code folder>/Experiment-[nnconfig]-[time]/]
-nsongs NSONGS            Number of songs to load [default: 99]
--doubleset               Double the dataset by taking a second extract from the audio files
                          [default: Disabled]
-ntests NTESTS            Number of external songs to test [default: 0]
-nval NVAL                % of training songs to use for validation after training [default: 5%]
-normvol NORMVOL          Normalise audio file volumes before STFT to given
                          absolute value (between -1 and 1) [default: Disabled]
-fs FS                    Sample rate of audio files in Hz [default: 44100]
--ffilter                 Apply band pass filter between 20 and 16,000 Hz
-alength ALENGTH          Length of audio in seconds [default: 12]
-aoffset AOFFSET          Offset of audio in seconds [default: 5]
-stftwindow STFTWINDOW
                          Number of samples per STFT window [default: 8192]
-scaler SCALER            Scale to normalise NN input; 0=[-1,1], 1=[0,1] [default: 0]
-nnconfig NNCONFIG        Configuration of the DRNN to train [default: 1]
--load                    Load data from pretrained NN for a specific configuration
--trial                   Short trial of the DRNN
--eval                    Only evaluate a DRNN, no training
-nLSTM NLSTM              Number of LSTM layers [default: 3]
-dcells DCELLS            Number of hidden units for drums NN [default: 200]
-bcells BCELLS            Number of hidden units for bass NN [default: 200]
-diters DITERS            Number of training iterations for drums NN [default: 200]
-biters BITERS            Number of training iterations for bass NN [default: 200]
-batchsize BATCHSIZE      Batch size when training [default: 8]
-earlystop EARLYSTOP      Patience value for early stop [default: No Early stop]
-afunction AFUNCTION      Activation function to use [default: tanh]
-l1 L1                    L1 recurrent regularises [default: 0.0]
-l2 L2                    L2 recurrent regulariser [default: 0.0]
-reducelr REDUCELR        Patience value for reduce LR on plateau [default: Disabled]
-lrate LRATE              Learning rate of the optimiser [default: 0.001]
--bnorm                   Apply batch normalitation between LSTM layers [default: Disabled]
--hps                     Use hps as input [default: Disabled]
```

An example of the syntax:

```
python SCSS.py -nnconfig 1 -bcells 100 -dcells 200 --ffilter -scaler 0 -biters 100 -diters 150 -ntests 3
-alength 12 -batchsize 8
```

3 Input Data Set

The data set provided includes the following files:

- Audio files of the bass tracks.
- Audio files of the drums tracks.
- Audio file with a mix of bass and drums.
- Test audio files with only the bass and drums mix.

There are a total of 99 available training data files and 31 test audio files.

To be able to evaluate the results it is required some sample files with the bass and drums files available separately; out of the 99 files, a 5% was left aside for evaluation purposes, leaving 95 files for training.

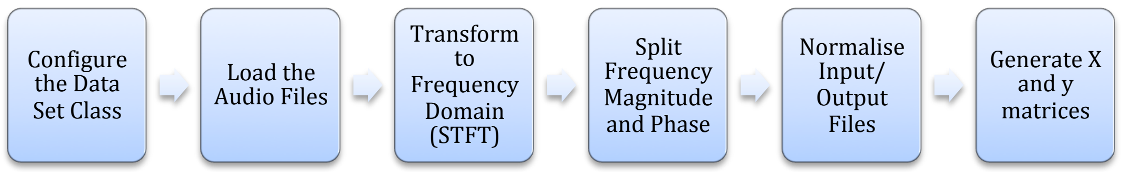
4 Data Preparation

As detailed in the research paper, the input of the neural networks will be in the frequency domain. It was decided to create a Python class to store all the configuration details to generate the data sets.

This class is called **AudioDataSet.py**

4.1 Data Load

The process followed to generate the data set is as follow:


Configure the Data Set Class
Load the Audio Files
Transform to Frequency Domain (STFT)
Split Frequency Magnitude and Phase
Normalise Input/ Output Files
Generate X and y matrices

The following optional parameters have been implemented:

- Apply a frequency band pass filter between 80 Hz and 15 KHz.
- Use harmonic percussive separation as data input instead a plain STFT.
- An option to double the data set by extracting 2 audio samples from each audio file.
- Add an offset to the audio files when loading them to avoid any initial silence in the audio track.
- Normalise the volume of all tracks to a set value (max 1.0)

All the experiments have been performed using audio files with an offset of 5 seconds and length of either 12 or 17 seconds.

4.2 Normalisation

In the research paper it only mentions that the input was normalised between -1 and 1. For reasons that will be discussed later, this normalisation didn't work well when training the model.

Several normalisations were tested during development – using the scaler function of the Sklearn library, the input was scaled and tested as:

- Data between -1 and 1:
- Data between 0 and 1.
- Data between 0 and 2.
- Data between -0.5 and 0.5
- Data equally distributed with a RobustScaler.

The challenge with the input data is that most of the information in the audio file has a very low frequency magnitude; this means that when scaling down, there will be data very close to the lower limit. This translates into the following depending on the scale used:

- [-1,1]: with this scaler most of the data will be very close to -1. This data would require an activation function that works well (and equally) with negative and positive values, leaving as best choice the hyperbolic tangent. However the problem is that with this activation function is sensible to the vanishing gradient problem, overall taking into account that the majority of the information will be closed to -1.
- [-0.5,0.5]: this option was used trying to increase the value of the gradient descent in the hyperbolic tangent by focusing the information in the middle of the activation function. This improved the results, but was discarded after testing the following scale.
- [0,1]: this scaling together with a ReLu activation function gave the best results among all the configurations.
- [0,2]: this scale was tested trying to give a greater margin to the input data; it was discarded, as the results were not much better than between 0 and 1.
- RobustScaler: this option was not fully tested, but the idea was to equally spread the information in the scale, instead using a proportional scale. However just because most of the frequency information has a low magnitude, does not mean that the most relevant information is there (noise will have very low magnitude and irrelevant). For this reason it was discarded.

After multiple experiments, it was decided to focus all the work in the original scale of the research paper [-1,1] and the scale that gave the best performance [0,1].

Something worth mentioning is that a lot of time was wasted during the project due to a bug in the code. The code was develop to have an input parameter to choose the activation function to use (tanh when negative values were used, ReLu for positive values only), however there was a hardcoded value on the activation function that was missed and made all the experiments that scaled [0,1] use the tanh activation function – the performance was bad and it created a lot of artifacts in the resulting audio files. The problem was identified by checking the range of the output data; first it was observed that the supposedly ReLu activation at the output was giving negative values (only a leaky ReLu would allow this); in the end the undesired variable assignment was found and removed.

4.3 Input Shape

The audio files are loaded from a wav file using the Librosa Python library. [`librosa.load()`] ; this returns the audio samples as an 1D array.

This array containing the audio data in the time domain needs to be converted into the frequency domain. Replicating the research paper, a Short Time Fourier Transform is used for this, with a Hanning window of 8192 and 50% overlap; the window size can be changed with an input parameter in the script, but all tests were performed with the values suggested by the research paper.

The STFT function in librosa returns a 2D array, with the frequency information in the first dimension and the window number in the second one. This dimensions are transposed to prepare the data for the LSTM neural network layer, which needs the step number (window) first and the information last.

4.4 Input Data Characteristics

As mentioned before, the input of the neural network is in the frequency domain. The STFT returns a matrix of complex numbers containing the frequency magnitude and the phase. Only the frequency magnitude is fed as input, using the phase at the end of the experiments to reconstruct the audio file. The function _complex_array() mixes back frequency and phase before applying the inverse STFT.

Another input for the neural network tested in some experiments is the harmonic percussive source separation available in Librosa. The function `librosa.decompose.hpss()` decomposes the STFT into harmonic (frequency change between frequency bands) and percussive (continuous frequency in time). This doubles the size of the input data, but should help the neural network identifying the right features for each instrument.

The output of the function is also a matrix of complex numbers, so it needs splitting in frequency magnitude and phase. In this case the phase information is discarded since the STFT phase information is used to reconstruct the audio file.

4.5 Populating the Training and Validation Data

To feed the data to Keras, the processed data is transformed from a list of arrays to a 3D array with the structure [number audio files, steps in time, frequency magnitude]. The array is called X_train. This is performed by the function `_populate_training_data()`

The output array is stored in the arrays y_train_bass and y_train_drums. Both are 2D arrays with the structure [audio file, step in time, frequency magnitude.

There is an equivalent matrix for the validation phase called X_val. The output is evaluated against the original tracks. The validation set is prepared in the function `_populate_validation_data()`

There is an extra function to prepare the validation data for the HPS data called The validation set is prepared in the function `_populate_validation_data_hps()`

4.6 From Prediction to Audio

The prediction of the neural networks will be the frequency magnitude of the bass and drums (2 separate networks).

The research paper shows the results by using the predictions to create a mask that is applied to the original audio file.

For completeness the Python code in this project outputs both the raw audio prediction and the masked audio.

To produce the raw prediction audio files, the steps below are followed:

- Scale back the data to its original values using the sklearn scaler.
- Reconstruct the complex number matrix by combining frequency magnitude and phase.
- Apply the inverse SFTF to the predicted file after readjusting the shape back to the structure [frequency, window].
- Optionally apply the band pass filter the same way as it may have been applied at the input files.
- Save to wav using the Librosa library.

These steps are executed in the function `_stft_to_wav(…)` called within the function `generate_tests(…)`
The mask version is generated following the research paper process:

- Calculate the masks for each instrument using the un-scaled frequency magnitude predicted by the neural networks.
- Apply the mask to the original audio file by multiplying the 2 matrices.
- Reconstruct the complex number matrix by combining frequency magnitude and phase.
- Apply ISTFT.
- Save to wav.

4.7 Test Files

Some mixed audio files are also provided to test the neural network. The process followed to transform from prediction to audio is equivalent to the where we have got the separate drums and bass tracks. However a new function was created to workaround some small differences:

- A new scaler is used that fits only each individual audio file.
- The resulting track is scaled back to the original data using this same scaler.
- The results cannot be evaluated using the evaluation measures, so it is only needed to save the predictions as audio files.

5 Neural Network Configuration

The neural network models are created using the library Keras with Tensorflow as the backend.

To make the code easier to read, this is split into 2 files:

- SCSS.py: main Python file – the network is created and trained in it.
- drnn.py: this file contains the different configurations (and definitions) of the neural networks used in the Configurations.

The Python library importlib is used to call the right configuration and return the model to SCSS.py, where it will be used.

The configuration to use can be given as an input parameter with –nnconfig. By default configuration 1 is used, which is the configuration detailed in the research paper.

Following the research paper, drums and bass are trained separately using 2 different neural networks. The loss function described on the research paper suggests that the optimisation is performed on the mean squared error of the error on the two networks, however it was confirmed that this is not the case and instead each network is optimised separately on their MSE.

5.1 Parameters

The following are parameters that can be tuned on all configurations:

- **Input shape**: this is automatically picked up based on the data generated for training.
- **Number of neurons per hidden layer**. The research paper states:
  - Bass: 100 neurons.
  - Drums: 200 neurons
- **Number of LSTM layers**: default is 3 as described in the research paper.
- **Activation function**: set to either ReLu or tanh, depending on the scaler.

- **L2 regularisation factor on recurrent layers**: default is 0.0 (no regularisation on weights)
- **Batch normalisation**: disabled by default.
- **Input type (hps):** needed to automate the desired number of output units; with hps the input is double the size of the output.
- **Learning rate**: default is 0.001 as described in the paper.

Other functionality implemented to make testing easier is:
- **Early Stop:** Usually used to avoid over-fitting while training the model. In this project it was used to save time while testing configurations; on the final experiments it was not used in the end.
- **Reduce Learning Rate on Plateau**: again trying to save time during model evaluation; this is a callback function in Keras that will reduce the learning rate of the optimiser after some epochs of no improvement on the validation loss. This means that at the beginning a high learning rate can be used to have larger changes with the gradient descent and when the loss does not improve this function will decrease the learning rate down to a minimum.

5.2 Configurations

There are a total of 10 configuration implemented in the code.

5.2.1 Configuration 1

This is the original research paper model.
- 3 time distributed dense layers (no activation)
- n LSTM layers (default 3)
- 1 time distributed dense layers with activation
- RMSProp optimiser with default values (γ=0.9, learning rate=0.001)

5.2.2 Configuration 2

In this configuration a fully connected layer is used at the input, which then feeds directly the LSTM layers. The output of the LSTM layers is then connected to another 2 dense layers (without activation function) and then to a fully connected layer with an activation function.
- 1 time distributed dense layers (no activation)
- n LSTM layers
- 2 time distributed dense layers (no activation)
- 1 time distributed dense layers with activation
- RMSProp optimiser with default values (γ=0.9, learning rate=0.001)

5.2.3 Configuration 3

This configuration is to test the performance when having the LSTM layers as input instead the dense layers.

- n LSTM layers
- 3 time distributed dense layers (no activation)
- 1 time distributed dense layers with activation
- RMSProp optimiser with default values (γ=0.9, learning rate=0.001)
- Optional: batch normalisation between LSTM layers, L2 recurrent regularization.

5.2.4 Configuration 4

This configuration adds a dropout layer at the input trying to mitigate the problem with the hyperbolic tangent – randomly dropping cells at the input will increase the error on the weights, so it could help increasing the magnitude of the gradient at back propagation.

- 1 Dropout layer with a probability of 0.2 dropout at input
- 3 time distributed dense layers (no activation)
- n LSTM layers
- 1 time distributed dense layers with activation
- RMSProp optimiser with default values (γ=0.9, learning rate=0.001)

5.2.5 Configuration 5

Same as the original configuration described in the research paper, but adding activation functions on all the dense layers at the input of the model.

- 3 time distributed dense layers with activation
- n LSTM layers
- 1 time distributed dense layers with activation
- RMSProp optimiser with default values (γ=0.9, learning rate=0.001)

5.2.6 Configuration 6

This configuration tests the dropout layer at the model output.

- 3 time distributed dense layers (no activation)
- n LSTM layers
- 1 time distributed dense layers with activation
- 1 Dropout layer with a probability of 0.2 dropout at output
- RMSProp optimiser with default values (γ=0.9, learning rate=0.001)

5.2.7 Configuration 7

In this case the dropout is performed in the recurrent cells in the LSTM layer.

- 3 time distributed dense layers with activation
- n LSTM layers with dropout in the recurrent cells of 0.2 probability
- 1 time distributed dense layers with activation
- RMSProp optimiser with default values (γ=0.9, learning rate=0.001)

5.2.8 Configuration 8

Perform the batch normalisation before the non-linear function in the recurrent layers as mentioned in the research paper by Sergey Ioffe et al[1].

- 3 time distributed dense layers (no activation)
- n LSTM layers with batch normalisation between LSTM and before the activation function
- 1 time distributed dense layers with activation
- RMSProp optimiser with default values (gamma=0.9, default learning rate=0.001)

5.2.9 Configuration 9

Use an Adam optimiser instead the RMSProp described in the research paper. The intention was evaluating the back propagation of the gradient descent with the hyperbolic tangent.

- 3 time distributed dense layers (no activation)
- n LSTM layers
- 1 time distributed dense layers with activation
- Adam optimiser with default values learning rate=0.001

5.2.10 Configuration 10

Implementation of and idea based on the research paper by Zhuo Li et al[2] with a few tweaks. This configuration applies a 2D convolutional neural network to the frequency spectrum of the STFT (as if it was an image); the kernel is width 1 to only convolute on each time step independently; height is the number of frequencies divided by 16, with a stride of 16 vertically to reduce the features to a number similar to the number of neurons of the hidden layers.

- 1 2D CNN with 1 filter, kernel size (1,No frequencies/16), strides (1,16)
- 1 time distributed dense layers (no activation)
- n LSTM layers
- 1 time distributed dense layers with activation
- RMSProp optimiser with default values (γ=0.9, learning rate=0.001)

[1] Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

[2] Deep Representation-Decoupling Neural Networks for Monaural Music Mixture Separation

5.2.11 Configuration 11

A smaller Kernel size with strides (1,1), so the output has a similar size to the input (only reduced by the kernel height, not by the strides).

- 1 2D CNN with 1 filter, kernel size (1,No frequencies/256), strides (1,1)
- 3 time distributed layer with activation
- n LSTM layers
- 1 time distributed layer with activation
- RMSProp optimiser with default values (γ=0.9, learning rate=0.001)

5.2.12 Configuration 12

For completeness, this configuration was designed to test the batch normalisation between all the layers in the model. It also adds an activation function after the batch normalisation.

- 1 time distributed dense layer
- Batch normalisation
- Activation function
- 1 time distributed dense layer
- Batch normalisation
- Activation function
- 1 time distributed dense layer
- Batch normalisation
- Activation function
- n LSTM layers
- 1 time distributed layer with activation
- RMSProp optimiser with default values (γ=0.9, learning rate=0.001)

6 Saving the Trained Model

The Python will automatically save the following the neural network models in the Keras format .h5

The sklearn scalers are also saved in files under the experiment folder – this could be useful to test the model without having to load the whole dataset again.

After a training session, the whole model (i.e. both model structure and weights) can be loaded from the experiment folder to either continue training or to predict new audios from other testing dataset.

7 Result Evaluation Metrics

7.1 Audio Metrics

The same evaluation metrics mentioned in the research paper have been used to measure the performance of the neural networks: SIR, SDR and SAR. However, SDR has been taken as the best reference since it measures the overall performance of the result:

$$SDR = 10\log\left(\frac{\|S_{tar}\|^2}{\|e_i + e_n + e_a\|^2}\right)$$

Being S_{tar} is the target separated source and e_i , e_n , e_a the interference, noise and artifacts error contribution.

The quality of the results is very dependent on the audio file. Out of the 4 tracks used on the network testing, 3 have been chosen as a reference. The evaluation metrics for each track has been averaged using a logarithmic average (metrics are in dB and a linear average would not reflect the actual results).

As an example, the calculation performed for the average SDR of 3 tracks is:

$$Average_{SDR} = 10\log\sum_{i=1}^{3} 10^{SDR_i/10}$$

However it is worth mentioning that the these metrics might not really reflect the subjective quality of the prediction perceived by the human ear.

The Python library used for these calculations is *mir_eval.separation.bss_eval_sources*

7.2 Training and Validation Errors

After each training iteration Keras returns the loss for both training and validation. These measures are saved in a matrix to then plot them with matplotlib and save them as png in the experiment folder.

The resulting errors obtained in the output are not as it would normally be expected. Ideally we should see a training error lower than the validation error, however this is not the case; a possible explanation for this is that the training error in Keras is the error on the last epoch of the iteration, whereas the validation error is an average of the result of all songs used for validation.

The percentage of train Vs validation set is specified when training the model; several values have been used during the experiments, but it was kept low due to make the most out of the limited number of tracks available for testing.

8 Hyperparameter Tuning

Hyper-parameters are any configuration set before the training. In this project there is a large list and due to the long time it takes to train a model, not all the desired settings could be tested.

The following is a list of some of these parameters:

- **Length of the audio file**: the longer the audio file, the more time steps there will be in the STFT and the more recurrent cells there will be in the LSTM layers. It was observed that the recurrent network performs better between 200 and 400 time steps, which was equivalent to between 12 and 17 seconds.
- **Size of the Hanning Window**: left to the default value of the research paper.
- **Learning Rate:** the original value in the paper is 0.001, however higher and lower values were tested.
- **Number of Layers:** most of the experiments use the required configuration, however it has been tested as well with 1 and 2 LSTM layers instead of 3.
- **Number of Neurons in Hidden Layers:** experiments include tests with 100, 200, 300, 400 and 241 (matching the output of the CNN network in experiment 10). This could also be tuned to have different number of neurons per layer.
- **Number of Iterations:** a minimum of 100 iterations were used for the bass and 150 for the drums. In some experiments it got to 300, which are some of the best results. This is very related to the model configuration – the more depth and neurons, the longer it may take to train.
- **Batch size:** most of the experiments were performed using a batch size of 8; batch 16 was tested but it didn't perform as well as 8. This parameter was set with RAM limitation and training time in mind. A smaller batch size doesn't need so much memory, but it will take longer to converge (and process).
- **Validation Split rate:** during the training the dataset is split and shuffled to avoid overfitting. The split rate is something else that could be tested and observed further.

9 Research Paper Results

The research paper information is the baseline to follow – the extract below shows the results presented in the research paper.

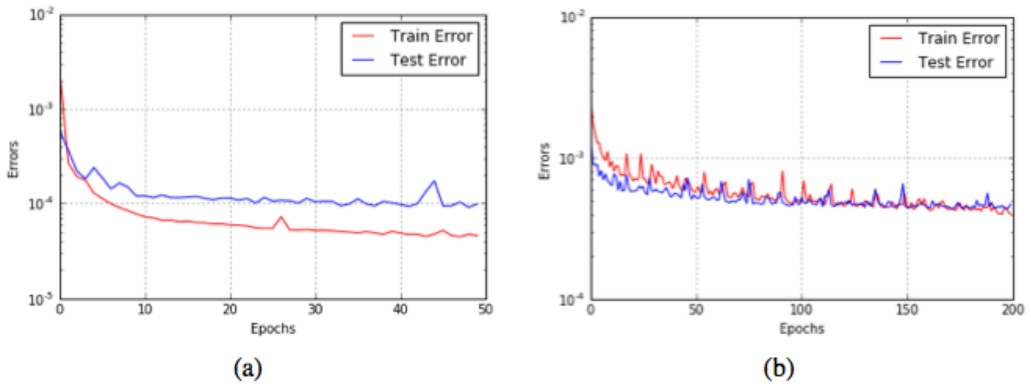
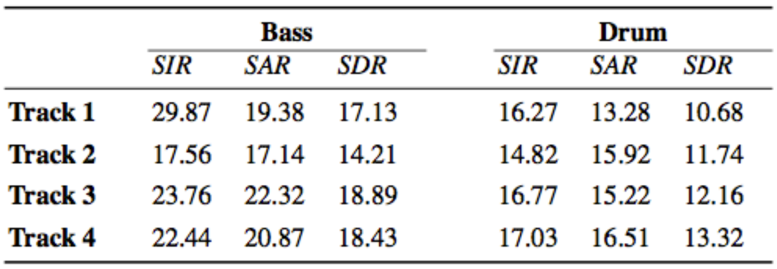


Fig. 7. Training and test errors for: bass (a) and drum (b) sources.



	Bass			**Drum**		
	SIR	*SAR*	*SDR*	*SIR*	*SAR*	*SDR*
Track 1	29.87	19.38	17.13	16.27	13.28	10.68
Track 2	17.56	17.14	14.21	14.82	15.92	11.74
Track 3	23.76	22.32	18.89	16.77	15.22	12.16
Track 4	22.44	20.87	18.43	17.03	16.51	13.32

Table 1. Results in terms of SIR, SAR and SDR of some test tracks for the proposed DRNN.

The aim of this project was getting the models to get as close as possible to a **training error of 5e-05 for the bass and 5e-04 for the drums**.

The magnitude of the training error will tell how well the model is able to learn.

The actual quality of the results will come from the evaluation metrics mentioned before. In the research paper the best **SDR** achieved was **18.89dB for the bass and 13.32dB for the drums**.

10 Experiment Results

10.1 General Remarks

Some generic concepts that apply to all the experiments:

- The dense layers are **time distributed**; this means that the same weights will be shared by all the time frames.
- The RMSProp optimiser in Keras uses as default configuration the values required by the research paper.
- A frequency filter between 80Hz and 15KHz was applied on all the experiments.

10.2 Research Paper Configuration

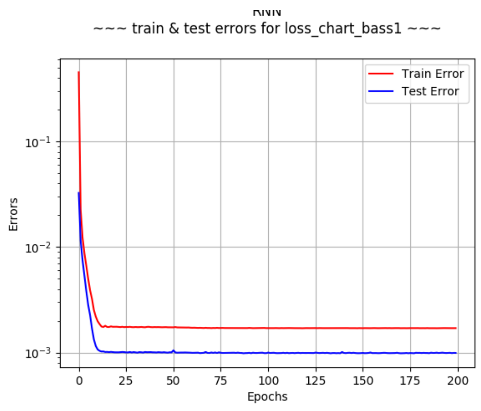
The research paper suggests the following model for the source separation:

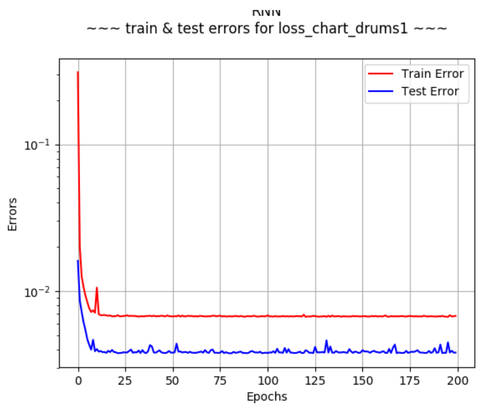
- 3 dense layers
- 3 LSTM layers
- An output (dense) layer
- 100 neurons in the hidden layers for the bass neural network.
- 200 neurons in the hidden layers for the drums neural network.
- Scaling the input between -1 and 1.
- RMSProp algorithm for the gradient descent optimisation with $\gamma = 0.9$ and a learning rate $\eta = 0.001$.

10.2.1 Results

The results of the presented configuration were not good in any of the experiments.

Below are the error values during training.





After very few epochs, neither of the models is able to learn.

The evaluation metrics are not good either:

	Drums			**Bass**		
	SDR	**SIR**	**SAR**	**SDR**	**SIR**	**SAR**
Track 1	-3.89	-3.72	15.51	7.15	7.41	20.26
Track 2	2.47	2.8	15.6	-0.22	-0.03	16.68
Track 3	8.33	9.2	16.26	-6.03	-5.94	17.88
Average	**3.90**	**4.17**	**17.73**	**4.26**	**5.04**	**17.00**

10.2.2 Issues Identified

10.2.2.1 Input Magnitude

The input data in the frequency domain had the majority of the values very close between 0 and 165, being the maximum value 1200.

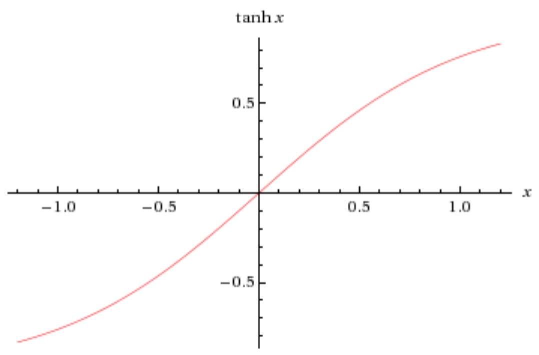
There were 2 tracks in the data set that were causing the 1200 maximum. After removing them the maximum value went down to 838, but still the majority of the data was 1/5th of the maximum value.

The table below represents the histogram of the input data.

0 to 167	**168 to 335**	**336 to 503**	**504 to 670**	**671 to 838**
97611693	10217	1164	205	37

Eventually in all the experiments all the dataset tracks were used, so the 1200 max value was present.

The main problem with this input and the research paper configuration is that when scaling between -1 and 1, the majority of the values will be close to -1. This is bad when using a hyperbolic tangent as activation function - the derivative at the extremes of the tanh will have a smaller gradient (below the graph of a this activation function plotted at Wolfram.com)



To the gradient issue we have to add as well the magnitude between values – if they are very small, the learning rate will be critical

To workaround this problem, different solutions were tested. One of them was applying a band pass filter to remove any meaningless information and hoping to remove any sub-bass frequencies that could be causing the peaks; the high values were indeed in the low frequency range, but too high to filter anything below it (around 150 Hz). Still a **band pass filter between 80Hz and 15 KHz** was applied in all experiments.

10.2.2.2 Learning Rate

Due to the fact that the values are scaled from [0,1200] to [-1,1], we are talking about very small changes when applying the gradient descent. This could mean that the proposed learning rate of 0.001 is too large for the model to learn (it will never converge further)

Another issue considered was the precision of the calculations (float32 vs float64). Float64 was used when possible to have as much precision as possible.

10.2.2.3 Overfitting

Another possible reason for the poor learning capacity of the model could be overfitting (the model is able to learn the training data but not generalise).

The training is performed with a validation split and shuffling – this should already help when overfitting.

Other alternative methods for regularisation that have been tested in the experiments are:

- Dropout (in different stages of the network).
- L1 and L2 regularisation.
- Batch normalisation.

10.2.2.4 Underfitting

The model detailed in the research paper should be able to learn the task at least as well as it is documented.

However, some other configurations have been tested aiming to improve the performance.

10.2.3 Alternatives

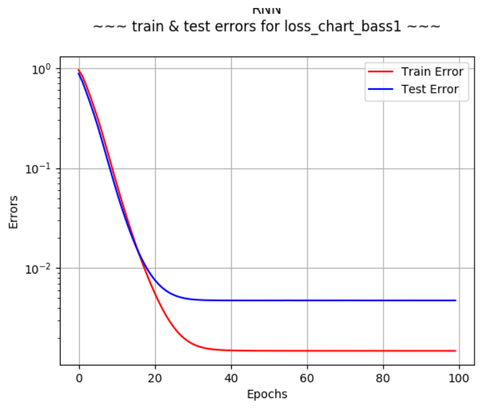
These are some other experiments with different hyperparameter tuning.

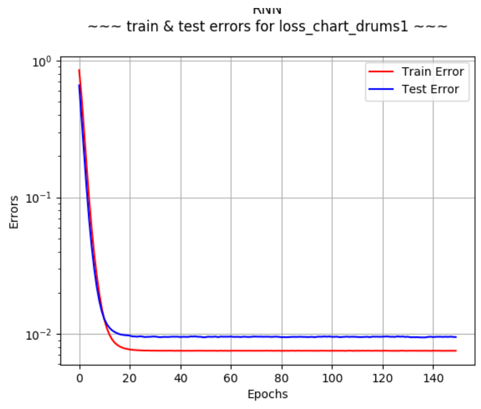
10.2.3.1 Lower Learning Rate

Using the same configuration, the learning rate was set to 0.0001 – the test error slightly increased, but the train error remained in a very similar value. The evaluation metrics showed a small improvement of 0.5dB on the drums and but 0.5dB worse performance on the bass.

Another change in this experiment is that it took longer to reach the same training error – this makes sense as the learning rate is much smaller and the changes on the weights will be smaller on every adjustment.

It should be noted that in this experiment the bass model was training with only 100 epochs and the drums with 150. The reason for lower epochs was mainly to save time, but training this configuration for longer would not increate the performance anyway, since the problem here is that the model is not able to minimise further the loss function.





	Drums			**Bass**		
	SDR	**SIR**	**SAR**	**SDR**	**SIR**	**SAR**
Track 1	-4.21	-4.07	16.23	8	8.3	20.42
Track 2	2.44	2.78	15.56	-0.11	0.09	16.21
Track 3	7.61	8.34	16.32	-5.45	-5.36	17.98
Average	**4.43**	**4.73**	**17.82**	**3.76**	**4.84**	**16.70**

With an even lower learning rate of 1e-05 there was a similar effect – drums performed better and bass worse.

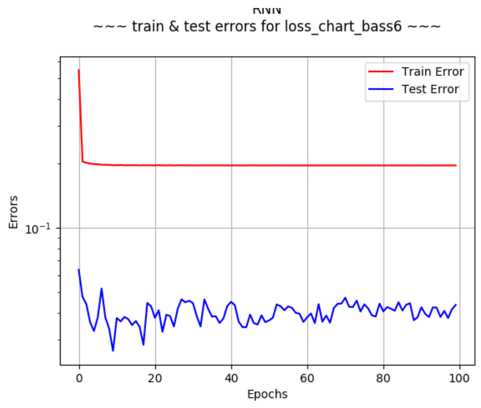
10.2.3.2 Dropout Regularisation

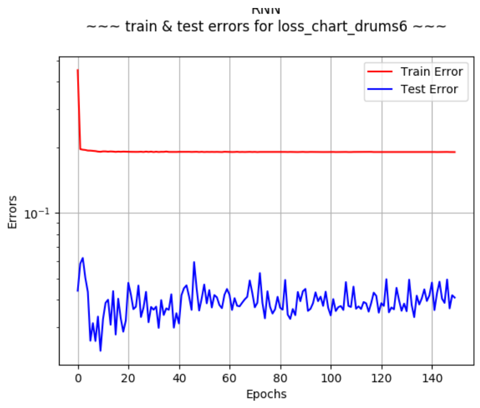
Different dropout configurations were tested, all of them with a 0.2 probability of dropout:

- Dropout at the input.
- Dropout at the recurrent cells.
- Dropout at the output.

Overall none of them performed better than the standard configuration, so dropout did not help as expected. The idea was to increase the weight error by dropping out some cells randomly and this way increase the gradient, but it did not help.

Based on the evaluation metrics, out of all 3 configurations, the one that performed better was the dropout on the output layer, achieving a similar performance to the standard configuration. However the error was around double than the first experiment.



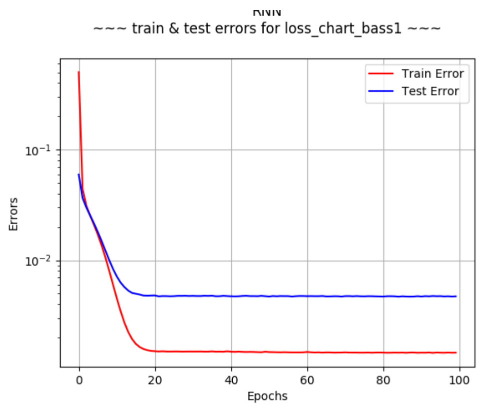


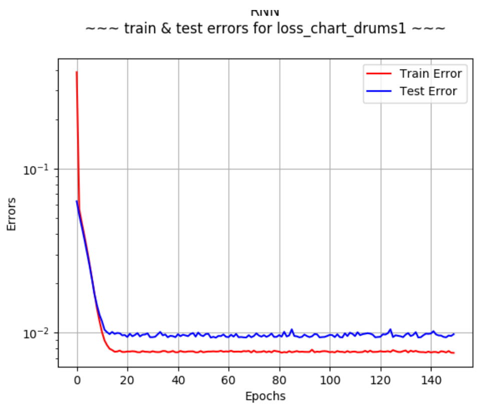
The training error is completely flat, however there is much more change on the test error – it was expected that Keras would ignore the dropout layer when validating the results, but based on the charts above it seems like it could be affecting the validation.

	Drums			**Bass**		
	SDR	**SIR**	**SAR**	**SDR**	**SIR**	**SAR**
Track 1	-4.69	-4.58	16.99	6.73	6.88	22.45
Track 2	3.44	4.01	14.06	0.2	0.32	18.59
Track 3	8.5	9.63	15.35	-5.89	-5.82	19.02
Average	**3.84**	**4.12**	**19.23**	**4.46**	**5.45**	**17.93**

10.2.3.3 L2 Regularisation

Something else that was studied during the tests was the size of the weights in the network. To do this, after each iteration of 10 epochs they were stored in a text file. They did not seem exploded but as an extra test the L2 norm regularisation was tested with a $\lambda = 0.0001$.
The loss function optimisation is very similar to lowering the learning rate, as well as the evaluation metrics. So again this did not help improving the learning.





	Drums			Bass		
	SDR	**SIR**	**SAR**	**SDR**	**SIR**	**SAR**
Track 1	-3.94	-3.71	14.27	7.81	8.04	21.18
Track 2	2.73	3.27	13.77	-0.2	-0.02	16.72
Track 3	7.71	8.96	14.26	-5.53	-5.45	18.24
Average	**4.43**	**4.73**	**17.82**	**3.76**	**4.84**	**16.70**

L1 regularisation was also tested in previous experiments, but the results were not good either and they are left out of the analysis.

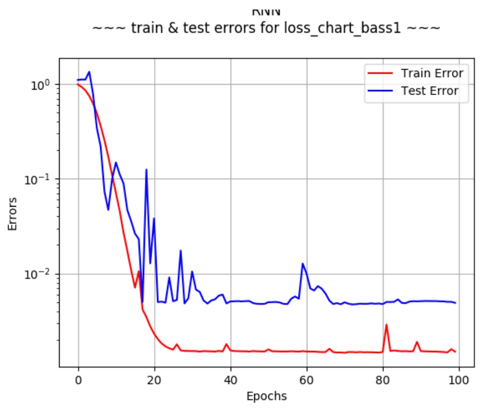
10.2.3.4 Batch Normalisation

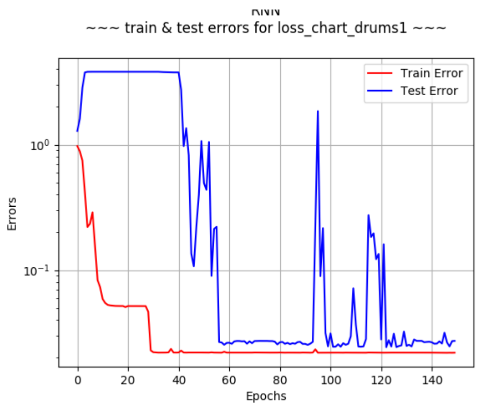
Taking into account that the amount of bass and drums may differ drastically between audio files, batch normalisation was used to try to keep the values between layers evenly spread in magnitude.

Three different experiments with batch normalisation were tested.

- Bath normalisation between the LSTM layers
- Batch normalisation bertween the LSTM layers, before the activation function (this experiment did not work).
- Bath normalisation between all layers in the model.

Using batch normalisation between the recurrent layers not only did not help, but also made the error minimisation very unstable.

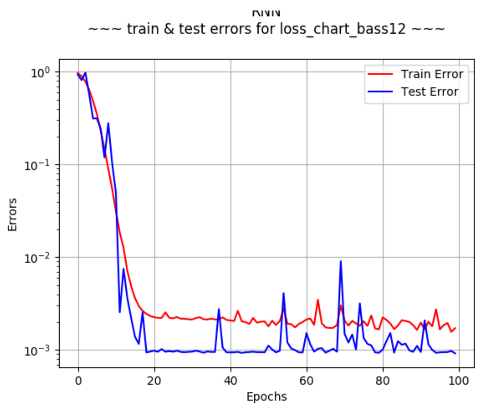


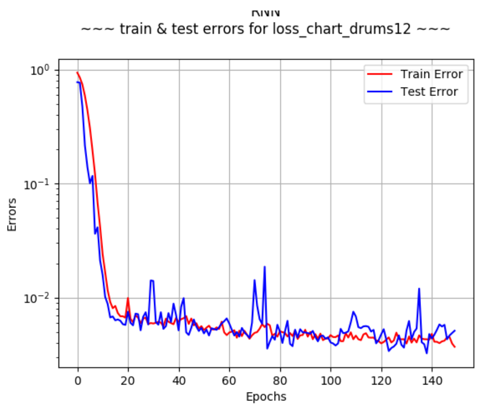


The source separation was also worse than the standard configuration. What it is interesting is to see a very high SAR (source to artifact ratio) on the bass predictions, but definitely not related to a better prediction of the bass.

	Drums			**Bass**		
	SDR	**SIR**	**SAR**	**SDR**	**SIR**	**SAR**
Track 1	-15.93	11.81	-15.64	7.17	7.18	34.19
Track 2	-14.74	9.13	-14.22	-0.98	-0.97	28.56
Track 3	-17.86	9.49	-17.38	-6.03	-6.02	29.85
Average	**2.45**	**9.79**	**29.42**	**-4.50**	**5.20**	**27.49**

On the other hand, when adding batch normalisation between the dense layers improved the bass error and it seems it made the weight updates more significant between iterations.





Unfortunately the predicted audio files were not better than the standard configuration

	Drums			**Bass**		
	SDR	**SIR**	**SAR**	**SDR**	**SIR**	**SAR**
Track 1	-4.44	-4.16	13.33	7.6	8.1	17.87
Track 2	2.86	3.28	14.87	0.89	1.45	12.42
Track 3	7.6	8.33	16.31	-4.84	-4.6	13.71
Average	**4.28**	**4.76**	**15.78**	**3.87**	**4.55**	**14.46**

It is appreciated that batch normalisation would require more time and experimentation. The configuration suggested in the original research paper about this topic did not work – the batch normalisation should be done before the activation function. Something else is that it was not applied in a timely distributed manner. It would be worth revisiting this option to fully understand the impact.

However, since the ReLu activation function gave much better results than any of the tanh experiments, this configuration was abandoned to focus on improving the performance with a ReLu configuration.

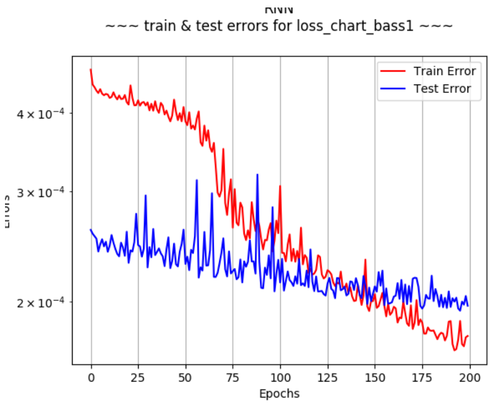
10.3 Research Paper Configuration with [0,1] and ReLu

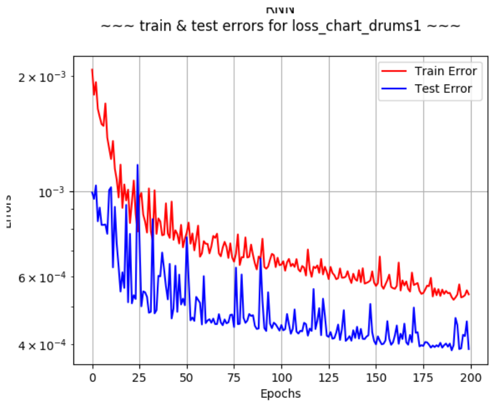
Normalising the input between 0 and 1 and using ReLu as activation function gave much better results.

The benefit of using ReLu is that the derivative is constant, so it is not affected by the gradient descent issue of the hyperbolic tangent.

This section presents a new baseline to take as a reference for the other experiments performed (all of them using [0,1] data normalisation and ReLu).

Regarding the error minimisation we can observe how with just changing to ReLu the learning of the model drastically improved. Now the error magnitude is lower than in the research paper and it seems it could have decreased further with more epochs.





The evaluation metrics are also better; looking at the average SDR, the drums neural network improved in 0.17dB and the bass 1.2dB. Even thought it might not seem like a lot, when playing the resulting audio tracks it can be perceived how better it is.

	Drums			**Bass**		
	SDR	**SIR**	**SAR**	**SDR**	**SIR**	**SAR**
Track 1	-2.42	-0.7	5.81	6.7	11.84	8.56
Track 2	3.82	7.99	6.56	1.69	4.5	6.23
Track 3	9.23	11.87	12.91	-1.64	0.26	5.74
Average	**4.07**	**8.74**	**7.14**	**5.46**	**8.08**	**9.62**

10.4 Other Configurations

This section details the results of multiple configurations tested for the task, sorted in order from best to worst performers. For the sorting the SDR logarithmic average of both bass and drums was for all 3 tracks was used.

HPS input was only used in 2 experiments – the data set generation as well as the training of the network take much longer. Unless stated, it should be assumed that the data input is the frequency magnitude of the STFT.

10.4.1 HPS Input - More Neurons – Less LSTM Layers

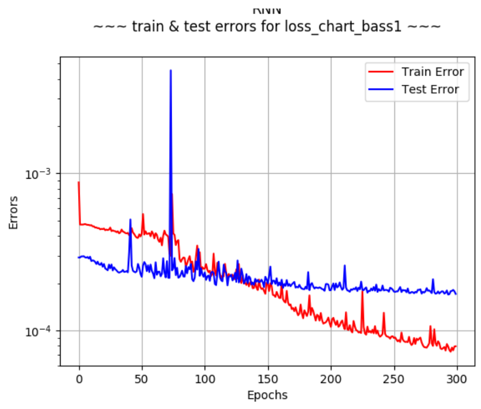
The best performer among all the experiments was the one with the following configuration:

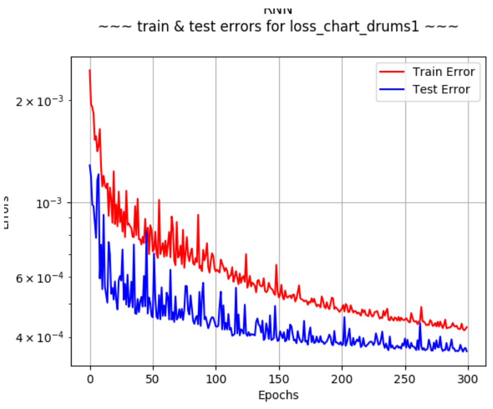
- Harmonic percussive separation as input.
- 400 neurons on all hidden layers for both drums and bass.
- 3 time distributed dense layers
- 2 LSTM hidden layers.
- 1 time distributed dense layer with ReLu activation function
- 300 iterations.
- Default optimiser.
- No regularisation on top of train/validation split and shuffle.

The main difference in this network is the data used in the input. With HPS we help the network by providing a different set of features split in harmonic (bass) and percussive (drums) information. This means that we are doubling the number of connections at the input, so the number of neurons in the hidden layers was also doubled.

The network was trained for longer because it was expected to need more time to train the higher amount of parameters.

As seen below, the error gets reduced to even lower the error in the research paper.





It can also be appreciated that the error could potentially have been reduced even further with more epochs.

Regarding the evaluation metrics, in average there is an improvement of 2.17dB in the drums NN and 4.4dB in the bass with respect to the first experiment (using tanh).

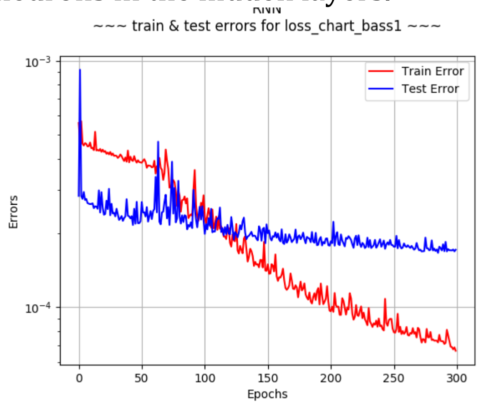
It also gets betters results compared to the standard configuration using ReLu.

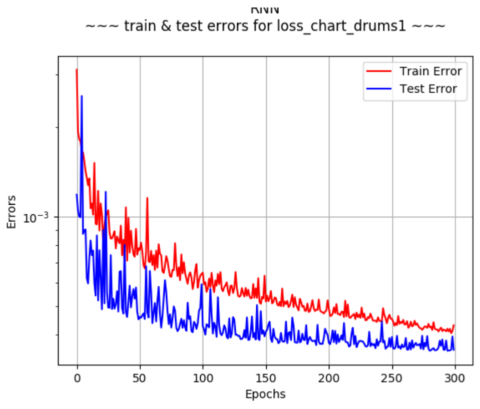
	Drums			**Bass**		
	SDR	**SIR**	**SAR**	**SDR**	**SIR**	**SAR**
Track 1	0.44	3.17	5.45	8.51	13.61	10.3
Track 2	5.96	10.36	8.29	4.3	7	8.44
Track 3	9.45	13.06	12.14	-1.03	1.07	5.64
Average	**6.07**	**10.78**	**8.44**	**6.12**	**9.46**	**9.54**

10.4.2 More Neurons – Less LSTM Layers

The next top performer was the same configuration as before, but using the standard STFT frequency magnitude as data input.

This means that maybe HPS is not worth the extra calculations and amount of memory needed during training and the important factor was the number of neurons in the hidden layers.





The evolution of the loss function optimisation is very similar to the previous experiment, although with slightly higher error in the drums model.

Other than that, the same observations apply.

	Drums			Bass		
	SDR	**SIR**	**SAR**	**SDR**	**SIR**	**SAR**
Track 1	1.72	5.77	4.92	8.92	12.81	11.41
Track 2	5.57	9.91	7.98	3.5	5.64	8.65
Track 3	8.63	12.61	11.08	-2.34	-0.66	5.97
Average	**6.33**	**10.37**	**8.89**	**5.28**	**8.80**	**9.05**

As before, results are similarly good compared to the results obtained with the baseline configuration.

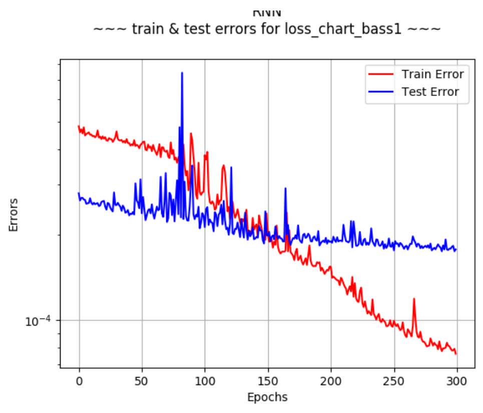
10.4.3 400 Neurons – 3 LSTM Layers

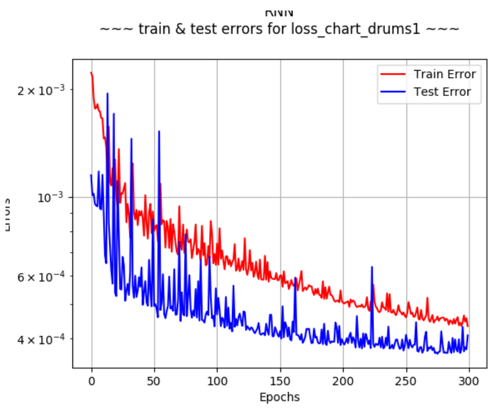
The next top performer configuration increases the number of LSTM hidden layers to 3.

An extra LSTM layer helped reducing further the error on the drums neural network. However it is a surprise that an extra layer does not help increasing the quality of the output result.

Since this network is deeper maybe with further training it could perform better, however due to the limited processing resources it was not tested.

The error optimisation charts are very similar to the first experiment (using HPS as input).





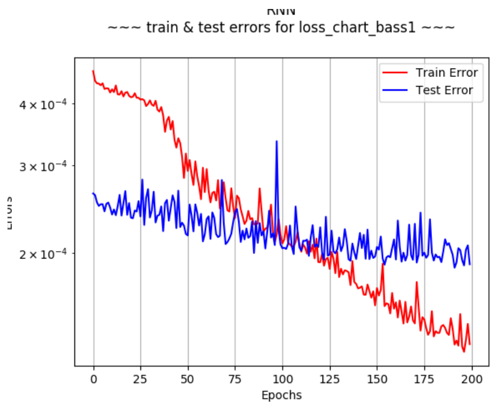
Below is the table with the evaluation metrics of this experiment.

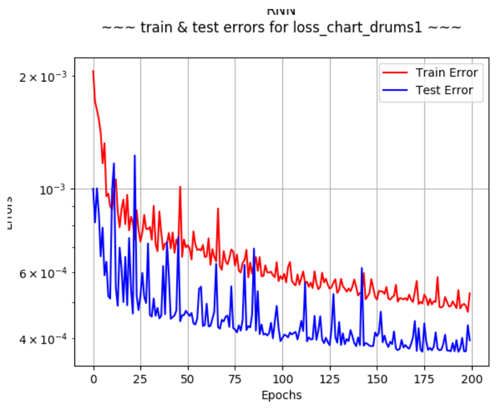
	Drums			**Bass**		
	SDR	**SIR**	**SAR**	**SDR**	**SIR**	**SAR**
Track 1	1.28	6.32	3.83	8.9	12.22	11.87
Track 2	5.46	11.72	6.91	3.2	5.08	8.9
Track 3	8.83	15.78	9.92	-1.9	-0.34	6.51
Average	**6.24**	**10.77**	**8.79**	**5.39**	**11.46**	**8.66**

10.4.4 Standard ReLu Configuration with 2 LSTM Layers

It is interesting to see how with 2 LSTM layer the network performed better than with 3.

The error got reduced slightly with respect to the standard ReLu configuration, probably because of less depth in the network.





The main difference can be seen in the evaluation metrics below, with a difference of 1.6dB in average:

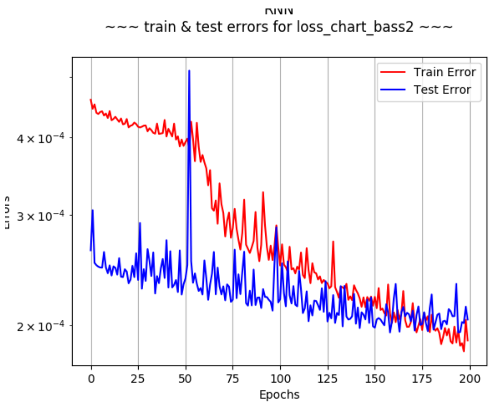
	Drums			**Bass**		
	SDR	**SIR**	**SAR**	**SDR**	**SIR**	**SAR**
Track 1	-1.01	1.24	5.36	7.77	12.17	9.98
Track 2	5.89	10.43	8.15	3.81	6.05	8.73
Track 3	8.63	12.11	11.48	-2.46	-1.04	6.66
Average	**5.51**	**9.83**	**8.22**	**5.34**	**8.47**	**9.41**

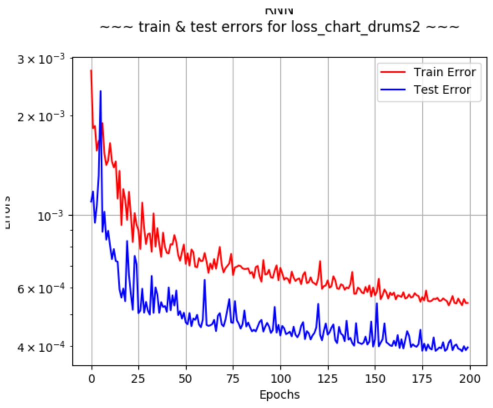
10.4.5 1 Dense – 3 LSTM – 3 Dense – 1 Dense with ReLu

The idea with this configuration was to have a single time distributed dense layer to process the input and reduce the number of connections and then pass it to the LSTM. To keep equivalent depth to the standard configuration the remaining 2 layers were added after the LSTM without an activation function, right before the last dense layer with ReLu activation function at the output.

Out of all the configurations with 3 LSTM layers and 100 (bass) 200 (drums) neurons, this is the one that perform the best.

The error is reduced similarly well to the best performing configurations:





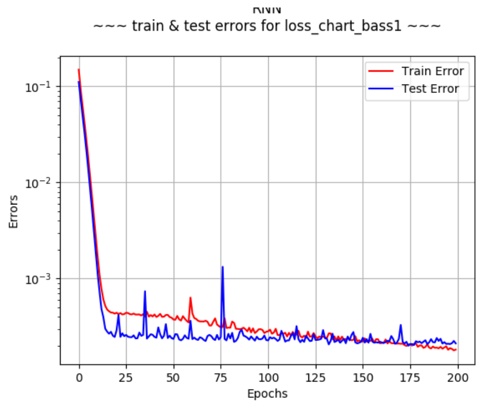
The performance is equivalent to having 2 LSTM layers only.

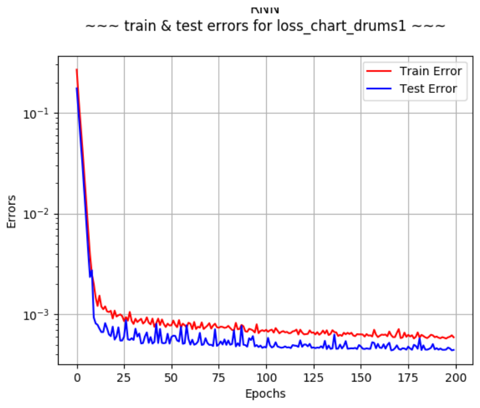
	Drums			**Bass**		
	SDR	**SIR**	**SAR**	**SDR**	**SIR**	**SAR**
Track 1	0.25	5.61	2.8	8.29	10.93	12.05
Track 2	4.73	9.02	7.26	2.52	4.36	8.5
Track 3	8.74	12.33	11.49	-2.28	-0.73	6.34
Average	**5.55**	**9.03**	**8.89**	**5.17**	**8.38**	**9.29**

10.4.6 2 LSTM layers with L2 regularisation

This experiment was intended to see the effect of the L2 regularisation. It was applied on the previous configuration with 2 LSTM layers.

The loss function is optimised in a more stable and constant way, however the error was not minimised as much as without the regularisation.





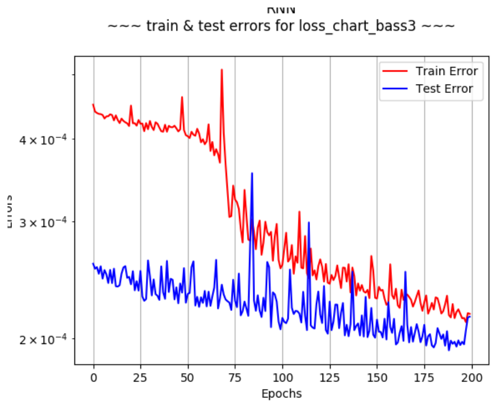
Consequently, using L2 regularisation slightly decreased the average performance. The drums neural network performed worse, although the bass neural network improved on two of the audio files.

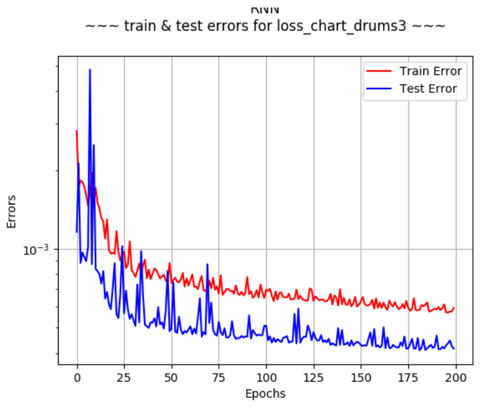
	Drums			**Bass**		
	SDR	**SIR**	**SAR**	**SDR**	**SIR**	**SAR**
Track 1	0.07	3.26	4.59	8.23	11.81	11.01
Track 2	5.21	9.25	7.88	3.26	5.21	8.83
Track 3	8.19	11.3	11.42	-3.15	-1.42	5.44
Average	**5.64**	**9.33**	**8.58**	**4.86**	**7.67**	**9.21**

10.4.7 LSTM Before Dense Layers

In this experiment the order of the layers was swapped, having LSTM before the time distributed dense layers.

Similar shape on the error minimisation on both models, with worse performance on the drums.



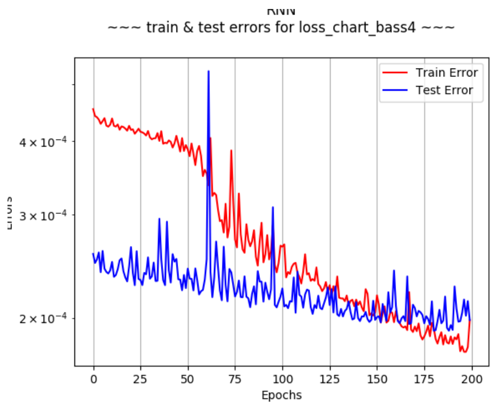


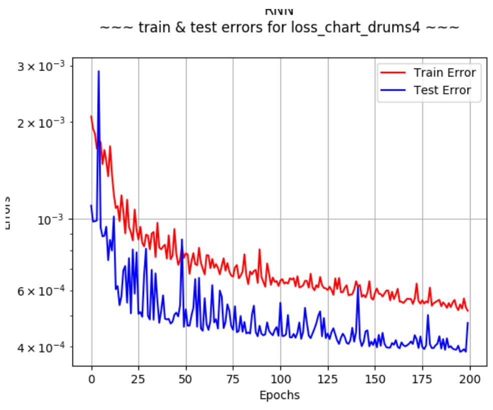
The result was better than the standard ReLu configuration,

	Drums			**Bass**		
	SDR	**SIR**	**SAR**	**SDR**	**SIR**	**SAR**
Track 1	1.04	7.67	2.78	8.22	9.52	14.58
Track 2	4.33	7.79	7.61	1.74	3.41	8.35
Track 3	8.4	10.7	12.63	-4.19	-3.44	8.88
Average	**5.49**	**8.41**	**10.84**	**4.67**	**6.81**	**10.40**

10.4.8 Standard ReLu Configuration with Input Dropout

This configuration adds a 0.2 probability dropout at the input. It did not affect much the results, with just a small improvement on the evaluation metrics compared to the standard ReLu configuration.





Evaluation metrics:

	Drums			**Bass**		
	SDR	**SIR**	**SAR**	**SDR**	**SIR**	**SAR**
Track 1	0.5	5.08	3.54	8.46	10.68	12.79
Track 2	4.56	8.67	7.25	2.15	3.58	9.22
Track 3	7.68	11.61	10.23	-3.46	-1.93	5.9
Average	**5.64**	**8.71**	**9.47**	**4.23**	**7.64**	**8.81**

10.4.9 CNN as Input Layer

Another configuration that was tested was adding a convolutional neural network as the first hidden layer.

This configuration could be expanded further, but the idea was treating the frequency spectrum as an image and applying a 2D convolutional layer to it; the goal was to find a good kernel that would help the following layers identify the bass and drums components.

Only 1 filter was used in the CNN due to computation constraints.

3 configurations were tried:

- **1x16 kernel with (1,1) strides followed by 1 dense layer**: this got the best result for the tanh baseline, however it was still not a good result compared to ReLu.
- **1x16 kernel with (1,1) strides followed by 3 dense layers**: a small improvement compared to just having a single dense layer after the CNN.
- **1x256 kernel with (1x16) strides** – the output of this layer is (number of frames)x241; this was feed to 241 neurons of the next hidden layer. It performed worse than the standard ReLu configuration, so the reduction of information with the 16 vertical stride did not work well.

10.4.10 Other Experiments

There are a total of 46 experiment results recorded, using all the configurations detailed in section 5.2. A table with all the results can be found on Appendix A

11 Summary

The performance of the original research paper was not achieved with any of the configurations.

The input normalisation cause issues in the training where the loss function could not be minimised as much as in the research paper and none of the regularisation techniques applied or different configurations helped to get the desired results.

Normalising the input between 0 and 1 and using ReLu activation function made a big different on the results and the source separation could be considered as good enough to see it as successful.

The best performing configurations were the ones with more neurons and it was an interesting result to see that a configuration with 2 LSTM hidden layers could perform better than with 3.

Something important observed during this project is how important is to provide a good input to the neural network. The data should be analysed thoroughly before any experimentation to save time.

12 Appendix A

Yellow: ReLu baseline – **Orange**: Tanh baseline

															Drums AVG			Bass AVG			AVG SDR
alength	bcells	biters	bnorm	dcells	diters	ffilter	hps	l2	lrate	nLSTM	nnconfig	ntests	scaler	batchsize	SDR	SIR	SAR	SDR	SIR	SAR	
17	400	300	False	400	300	True	True	0	0.001	2	1	5	1	8	6.07	10.78	8.44	6.12	9.46	9.54	6.10
17	400	300	False	400	300	True	False	0	0.001	2	1	5	1	8	6.33	10.37	8.89	5.28	8.80	9.05	5.84
17	400	300	False	400	300	True	False	0	0.001	3	1	5	1	8	6.24	10.77	8.79	5.39	11.46	8.66	5.83
17	200	300	False	200	300	True	False	0	0.001	2	1	5	1	8	5.91	10.46	8.25	5.10	10.11	8.41	5.52
17	100	200	False	200	200	True	False	0	0.001	2	1	3	1	8	5.51	9.83	8.22	5.34	8.47	9.41	5.43
17	300	300	False	300	300	True	False	0	0.001	2	1	5	1	8	6.02	9.95	8.82	4.71	7.52	8.83	5.42
17	100	200	False	200	200	True	False	0	0.001	3	2	3	1	8	5.55	9.03	8.89	5.17	8.38	9.29	5.36
17	100	200	False	200	200	True	False	0.001	0.001	2	1	3	1	8	5.64	9.33	8.58	4.86	7.67	9.21	5.27
12	241	200	False	241	200	True	False	0	0.001	3	1	3	1	8	5.80	9.07	9.29	4.58	8.59	8.73	5.23
17	100	200	False	200	200	True	False	0	0.001	3	3	3	1	8	5.49	8.41	10.84	4.67	6.81	10.40	5.10
17	100	200	False	200	200	True	False	0	0.001	3	4	3	1	8	5.64	8.71	9.47	4.23	7.64	8.81	4.99
17	100	300	False	100	300	True	False	0	0.001	2	1	5	1	8	5.21	9.23	7.98	4.58	7.09	9.15	4.91
12	241	200	False	241	200	True	False	0	0.001	3	1	3	1	4	5.17	11.24	7.33	4.45	7.45	8.68	4.82
17	100	200	False	200	200	True	False	0	0.001	3	1	3	1	8	4.07	8.74	7.14	5.46	8.08	9.62	4.82
17	100	200	False	100	200	True	False	0	0.001	2	1	3	1	8	4.90	9.58	7.29	4.60	7.92	8.60	4.75
17	100	200	False	100	200	True	False	0	0.001	3	1	3	1	8	4.81	8.21	9.05	4.56	8.52	7.90	4.69
17	100	200	False	200	200	True	False	0	0.001	3	7	3	1	8	5.44	8.04	11.40	3.56	6.27	9.57	4.60
17	100	200	False	200	200	True	True	0	0.001	3	1	3	1	8	4.39	9.00	7.03	4.66	8.17	8.76	4.53
17	100	200	False	200	200	True	False	0	0.001	3	9	3	1	8	4.46	7.80	7.88	4.22	9.89	8.20	4.34
12	100	200	False	200	200	True	False	0	0.001	3	10	3	0	8	4.66	5.29	15.85	3.88	4.53	15.30	4.29
12	241	200	False	241	200	True	False	0	0.001	3	10	3	1	8	5.40	8.86	10.08	2.74	5.21	7.63	4.27
17	100	200	False	200	200	False	False	0	0.001	3	1	3	1	8	4.35	7.19	8.38	4.14	7.16	8.86	4.25
17	200	200	False	200	200	True	False	0	0.001	3	1	3	1	8	4.64	9.26	7.03	3.79	7.15	7.35	4.23
17	100	100	False	200	150	True	False	0	0.001	3	6	3	0	8	3.84	4.12	19.23	4.46	5.45	17.93	4.16
17	100	200	False	200	200	True	False	0	0.001	3	6	3	1	8	4.73	8.21	8.70	3.43	6.99	8.50	4.13
17	100	200	False	100	200	False	False	0	0.001	2	1	3	1	8	5.03	8.64	8.14	2.95	8.67	8.35	4.11
12	100	100	False	200	150	True	False	0	0.0001	3	1	3	0	8	4.49	4.79	17.97	3.70	4.33	16.92	4.11
12	100	100	False	200	150	True	False	0.0001	0.001	3	1	3	0	8	4.43	4.73	17.82	3.76	4.84	16.70	4.11

															Drums AVG			Bass AVG			AVG SDR
alength	bcells	biters	bnorm	dcells	diters	ffilter	hps	l2	lrate	nLSTM	nnconfig	ntests	scaler	batchsize	SDR	SIR	SAR	SDR	SIR	SAR	
17	100	200	False	200	200	True	False	0	0.001	3	1	3	0	8	3.90	4.17	17.73	4.26	5.04	17.00	4.09
12	100	100	True	200	150	True	False	0	0.001	3	12	3	0	8	4.28	4.76	15.78	3.87	4.55	14.46	4.08
12	100	100	False	200	150	True	False	0.001	0.001	3	1	3	0	8	4.27	4.58	20.07	3.81	5.15	14.01	4.04
12	241	200	False	241	200	True	False	0	0.001	3	10	3	1	16	4.73	6.94	9.55	3.15	5.94	8.31	4.01
12	241	200	False	241	200	True	False	0	0.001	3	1	3	1	16	4.47	8.53	7.24	3.45	6.50	9.14	3.99
12	100	100	False	200	150	True	False	0	1.00E-05	3	1	3	0	8	4.42	5.05	26.09	3.45	6.01	24.00	3.96
17	100	100	False	200	150	True	False	0	0.001	3	4	3	0	8	3.83	4.16	17.02	4.06	4.85	16.15	3.94
17	100	200	False	100	200	False	False	0	0.001	2	1	3	1	8	4.74	8.84	7.40	2.95	7.67	7.07	3.94
12	100	200	False	200	200	True	False	0	0.001	3	11	3	1	8	4.31	7.49	7.94	3.41	6.68	8.11	3.89
17	100	100	False	200	150	True	False	0	0.001	3	7	3	0	8	3.63	3.97	16.87	3.93	4.84	16.21	3.79
17	400	300	False	400	300	True	False	0	0.001	3	1	5	0	8	3.58	3.91	16.81	3.76	4.71	15.82	3.67
12	100	100	False	200	150	True	False	0	0.001	3	1	3	0	8	4.08	4.50	16.27	2.94	3.73	15.35	3.55
17	100	200	False	200	200	False	False	0	0.001	2	1	3	1	8	4.66	8.41	7.60	1.95	8.65	6.00	3.51
12	100	200	False	200	200	True	False	0	0.001	3	10	3	1	8	3.94	6.59	8.93	2.67	7.14	7.11	3.35
17	100	200	False	100	200	False	False	0	0.001	3	1	3	1	8	2.76	9.22	7.63	3.70	6.56	8.10	3.25
17	100	200	True	200	200	True	False	0	0.001	3	8	3	1	8	2.47	3.60	10.24	1.56	3.57	7.55	2.04
17	100	200	False	200	200	True	False	0	0.001	3	5	3	1	8	1.12	9.21	5.61	1.42	5.17	5.25	1.27
12	100	100	True	200	150	True	False	0	0.001	3	1	3	0	8	2.45	9.79	29.42	-4.50	5.20	27.49	0.24
17	100	200	True	200	200	True	False	0	0.001	3	1	3	1	8	-2.91	12.77	6.90	-3.87	6.23	8.53	-3.36