Classification and separation of musical instruments

Ian McKenzie, Jakob Hornum, Luc Voorend, Sascha Brown & Simon Ørgaard

Niels Bohr Institute
June 2024

UNIVERSITY OF COPENHAGEN

Goals of the project

Use ML algorithms to classify and separate musical instruments from audio files

Three stage project:

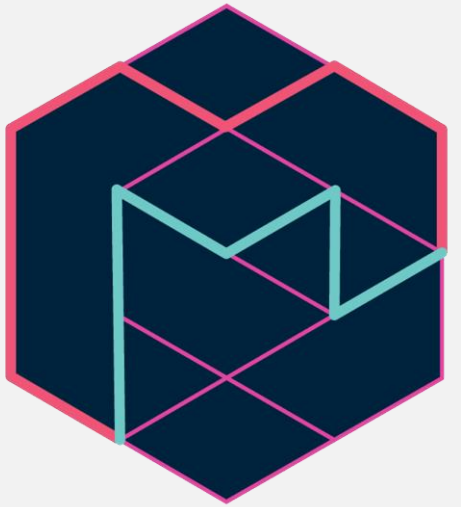
1) Single instrument classification
2) Multi instrument classification
3) Instrument separation

Applied methods

- Split up our efforts to work on many different models
- Mixed successful / not successful
- This presentation will give an overview of the models:
  - From easy to more complex
  - Describing the performance of each
  - Explaining the pitfalls and challenges of each
- More details and link to GitHub with full code in the appendix

Nsynth dataset

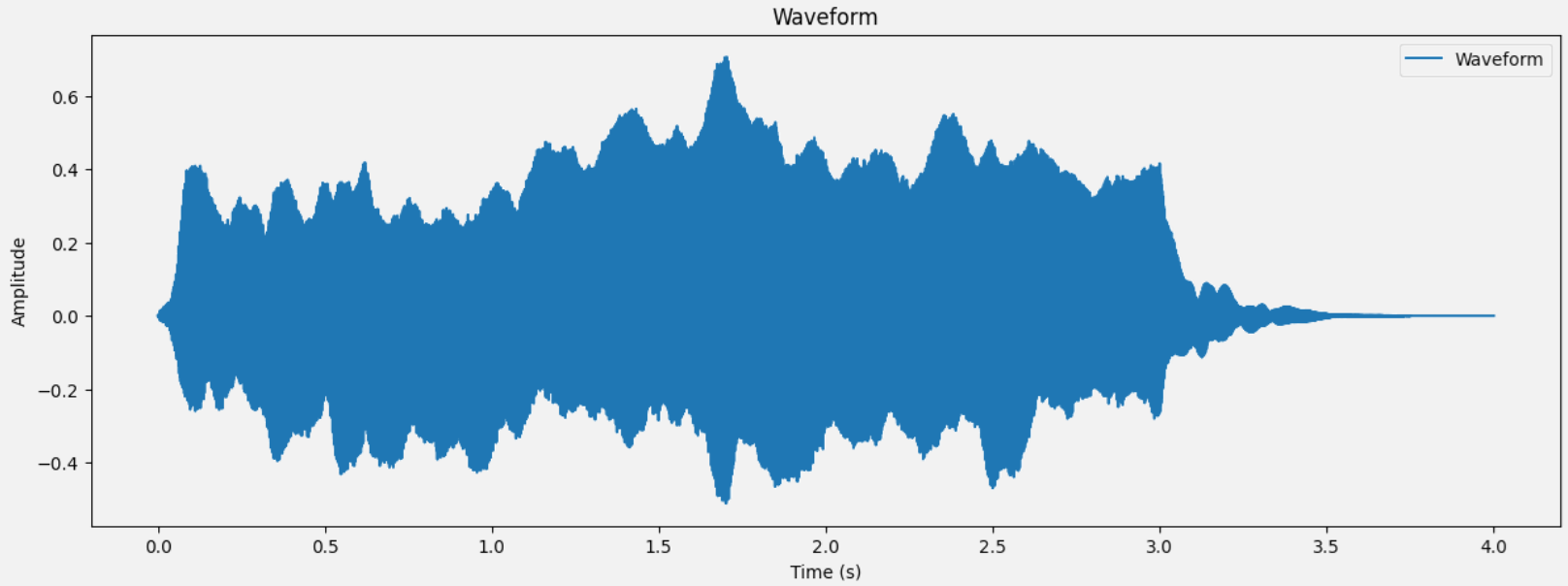
- Attempt for a MNIST like dataset for audio
- 305,979 musical notes - each with unique pitch, envelope and timbre.
- 4 second audio file per musical note
- Many features and labels available for each sample
    - We only used the instrument type



Source:
Jesse Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Douglas Eck, Karen Simonyan, and Mohammad Norouzi. "Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders." 2017. https://doi.org/10.48550/arXiv.1704.01279

What does music look like?

- Waveform data:

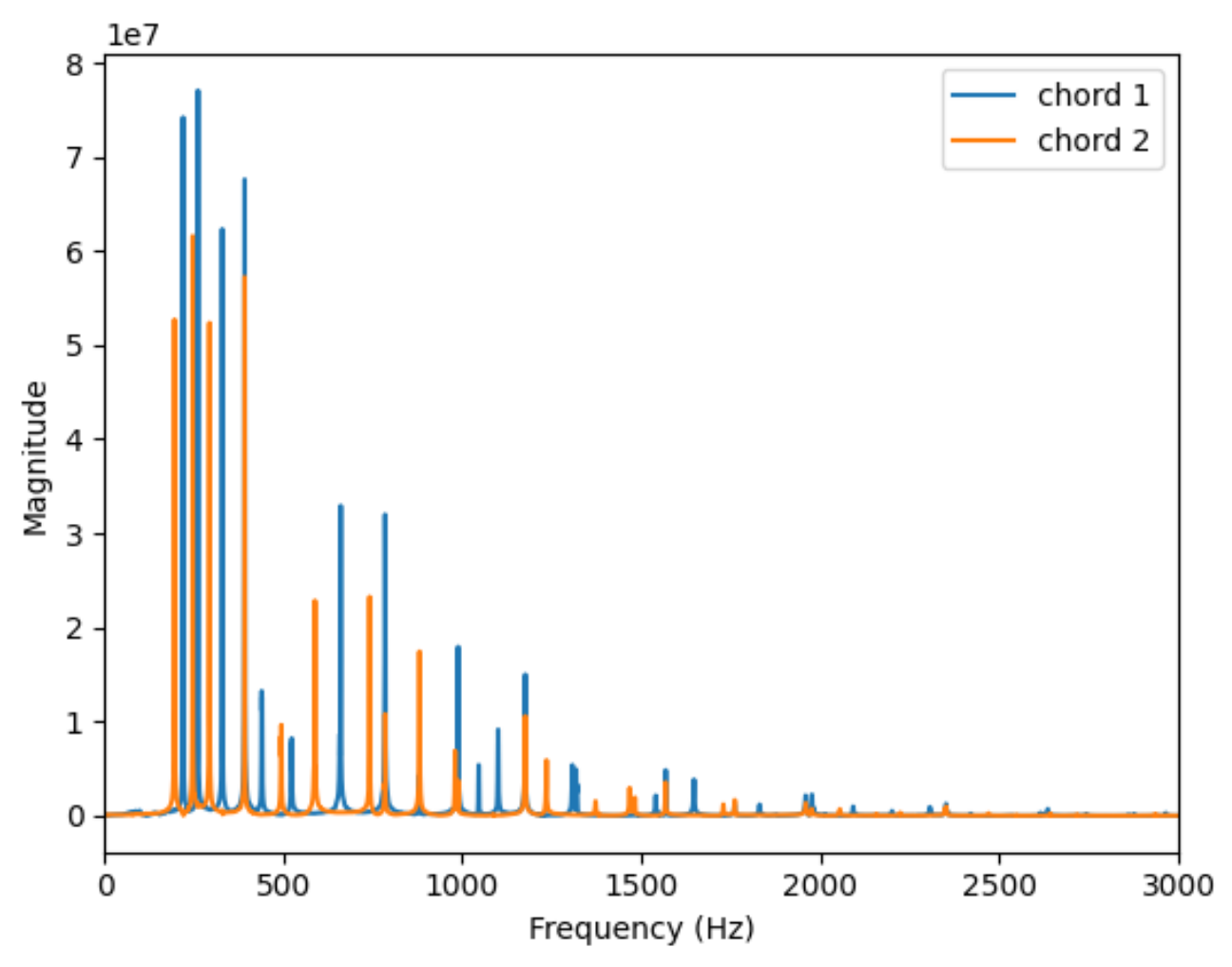


- Contains important features in a huge range of time scales:
- (20 Hz to 20kHz) ~ (0.05 seconds to 50 microseconds)
- And much larger time scales for envelope features like attack, sustain, etc.

A clearer picture is seen in Fourier transform

Harmonics play a central role in giving each instrument a unique sound profile.

But in a song, the melody and chords are constantly changing in time, which would clutter this plot to the right.

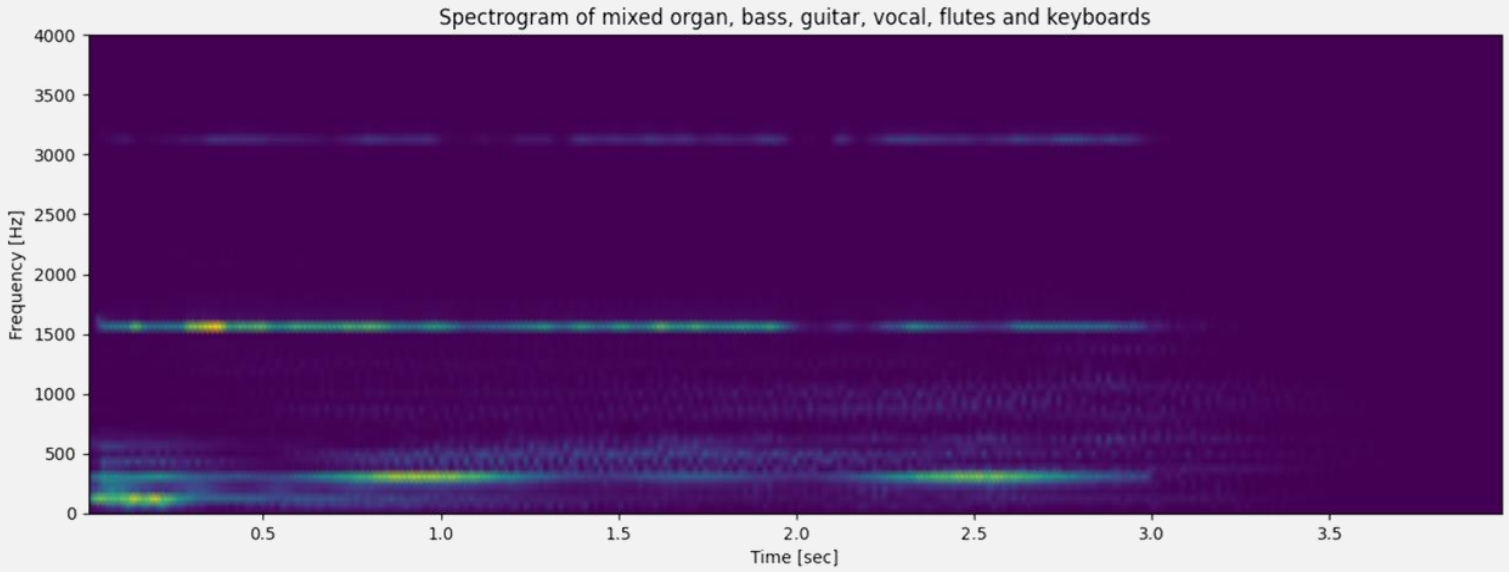


Spectrograms

- Perhaps the clearest way to represent sound data.

- Calculated using the "Short time Fourier transform" on the waveform and stacking the timed data.

$$: \int_{-\infty}^{\infty} x(t)w(t-\tau)e^{-i\omega t}\,dt$$



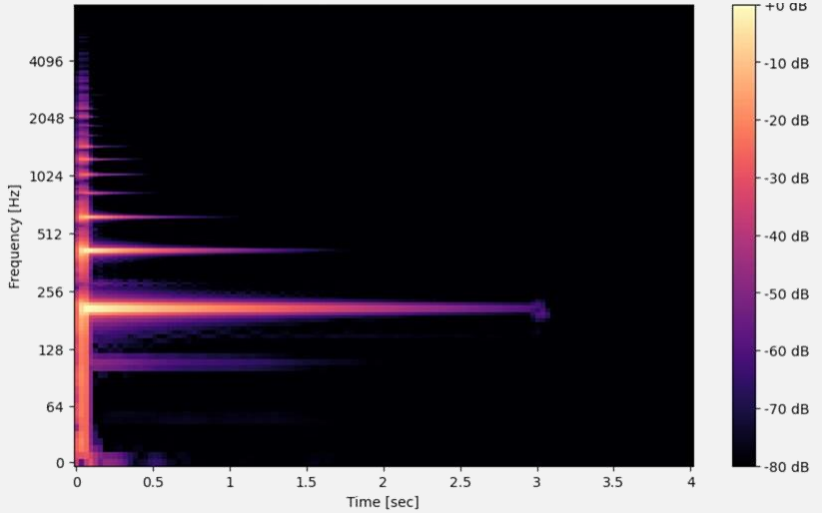
Single instrument classification

The goal:

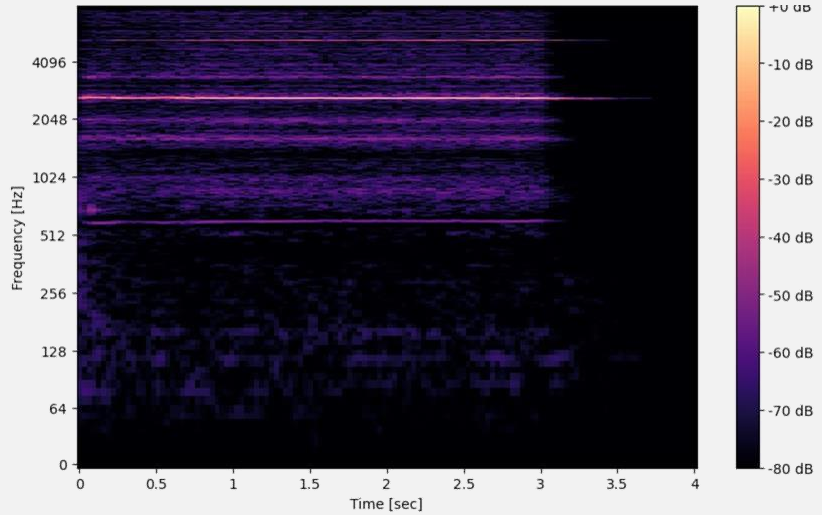
Classifying the instrument type from an audio file containing sound from a single instrument

Loss to minimize:

$$multi{:}softmax = \frac{\#(wrong\ cases)}{\#(all\ cases)}$$



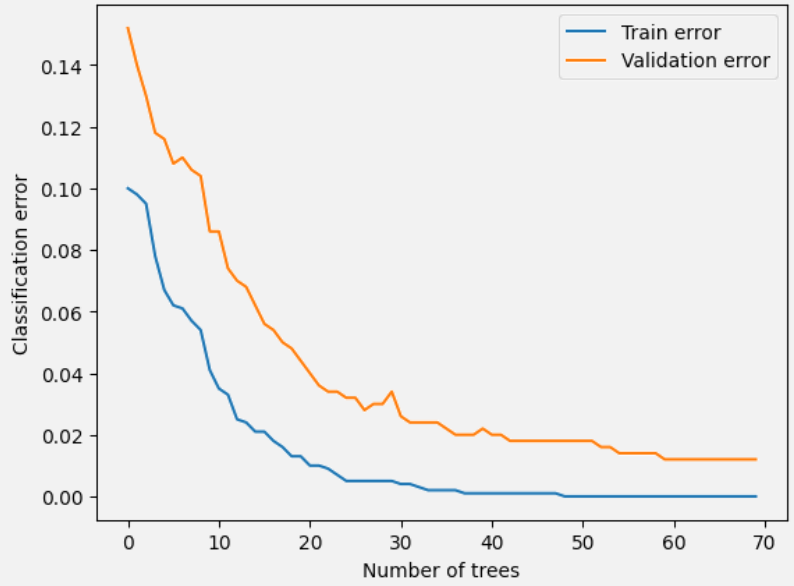
Bass

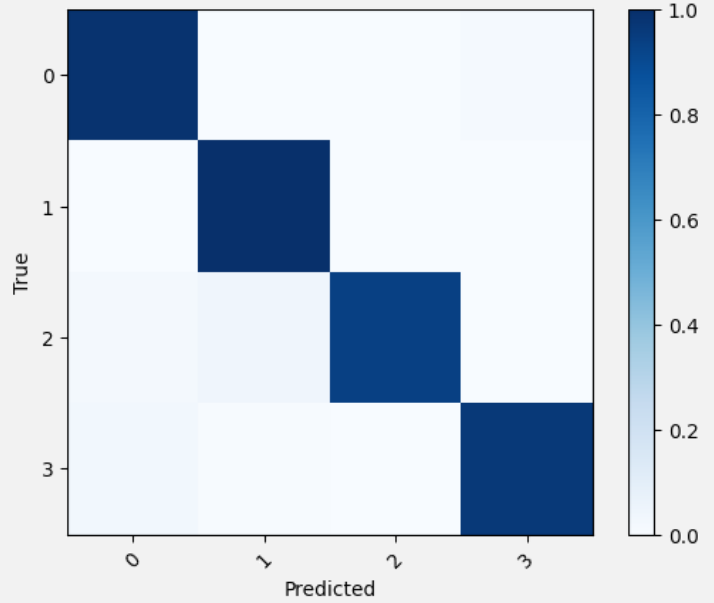


Flute

XGBoost classifier

- Flattening the spectrogram to a 1D array turned out to be enough
- Boosted decision tree is simple and fast
- Trained on 4000 samples (4 possible instruments)
- 98% accuracy in the testing set

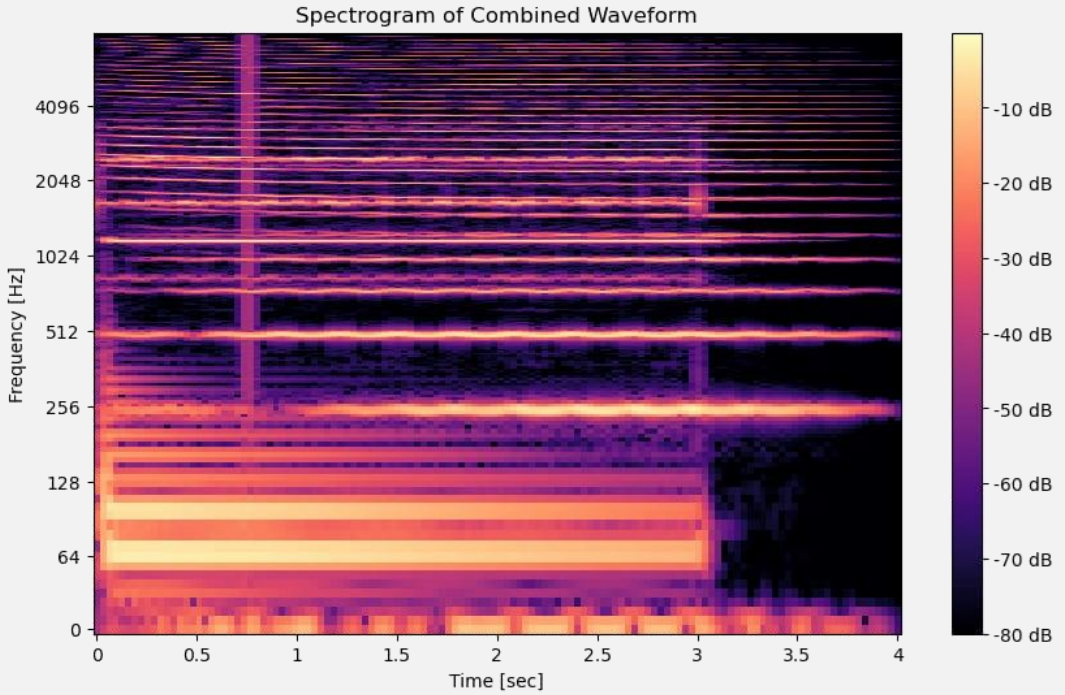




Multi instrument classification

The goal:

Predicting the presence/absence of 6 instruments from an audio file containing sound from a various amount of instruments (1 to 6)

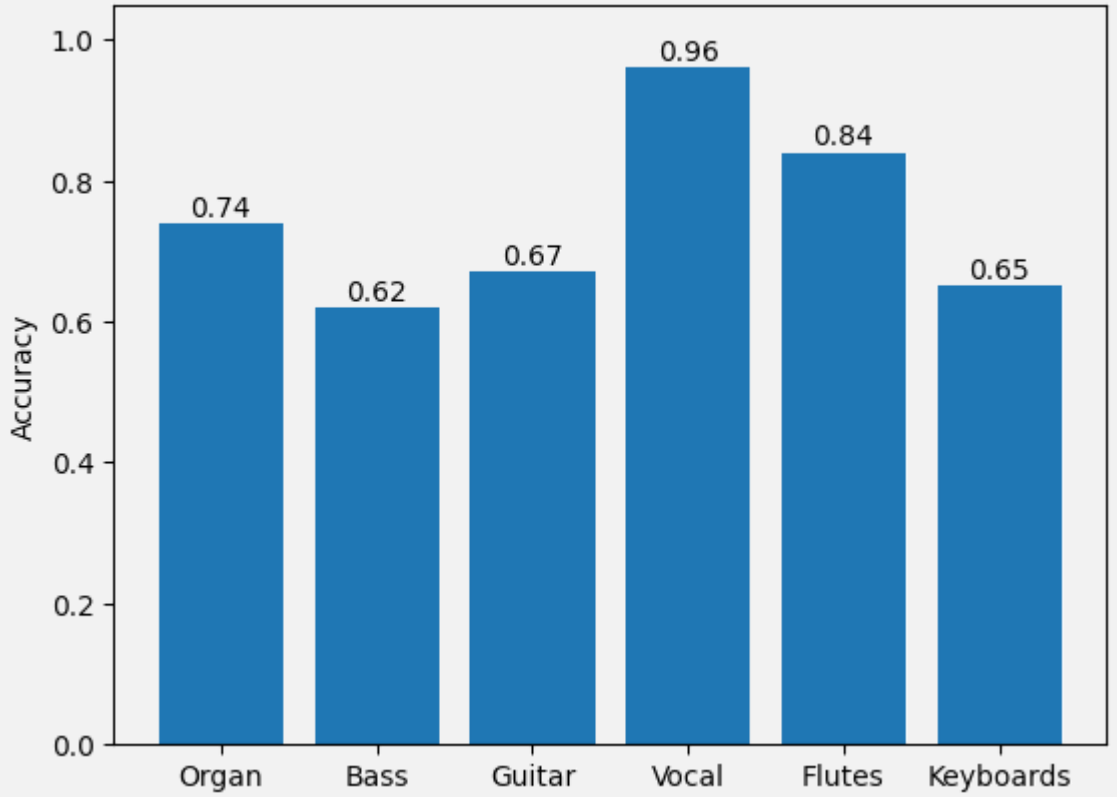


Loss to minimize:

$$\frac{\sum_{i=1}^{N}(|\hat{y}_i - yi|)}{N}$$

XGBoost classificier

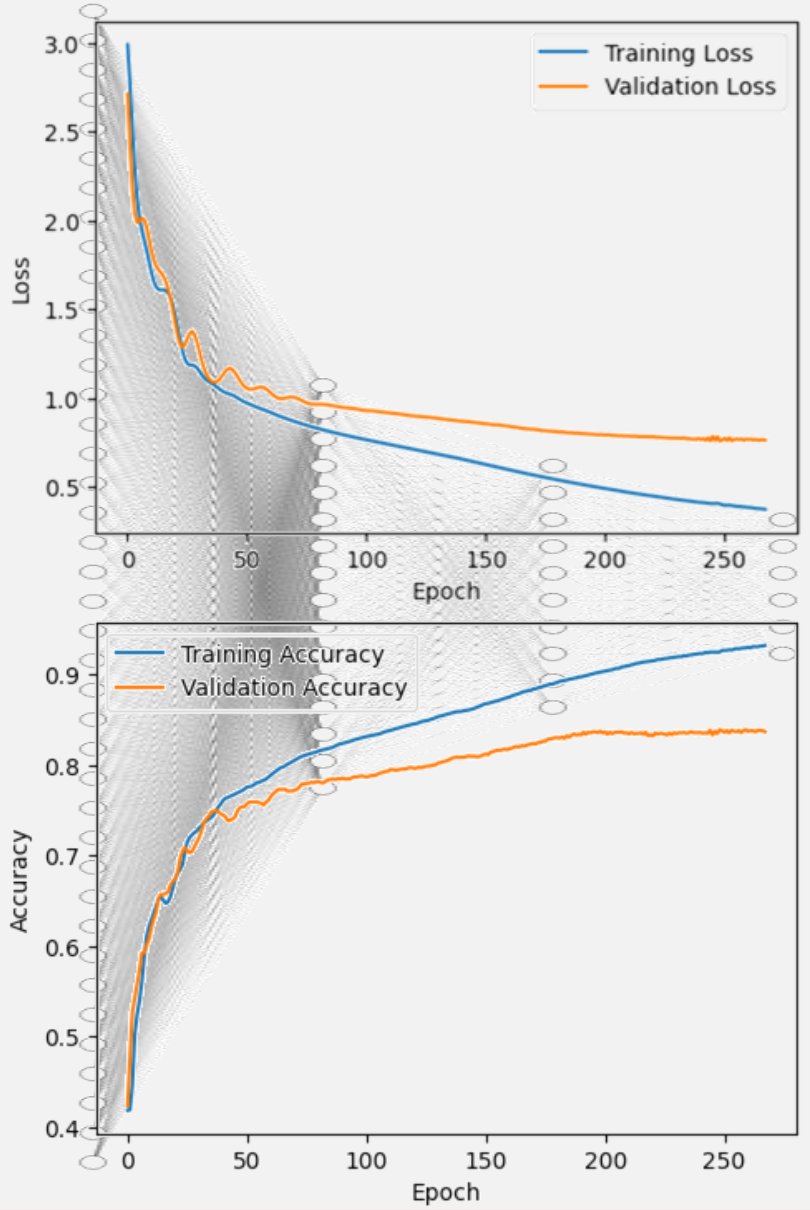
- Why not try to see if it works?
- Trained similarly on 4000 samples
- 6 individual models trained on ‘noisy’ data
- Total accuracy of 75% on test set



Multi layer perceptron

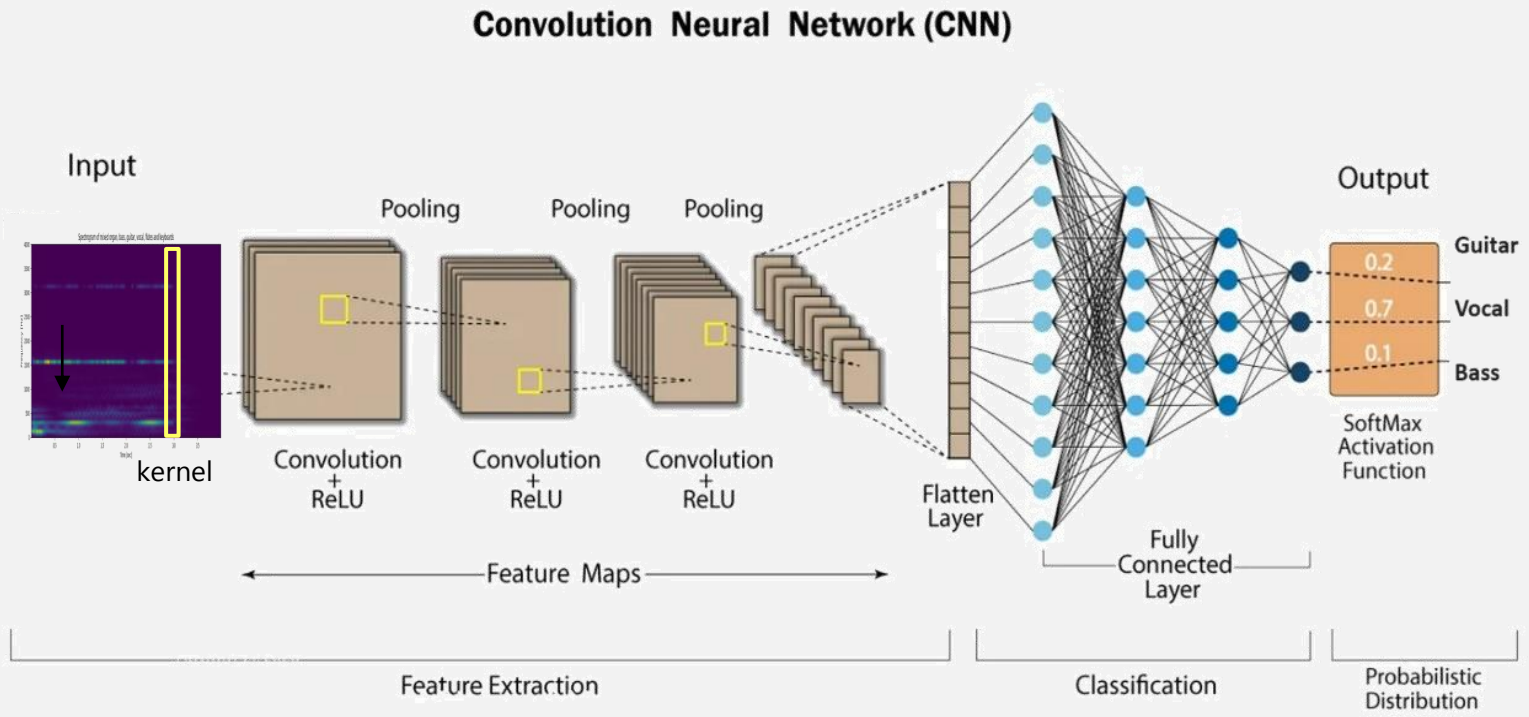
- Model architecture:
  - 2 hidden layers
  - ReLu activation functions
  - Non-optimal shape, but fast

- Trained on 4800 samples
- Total accuracy of 85% on test set



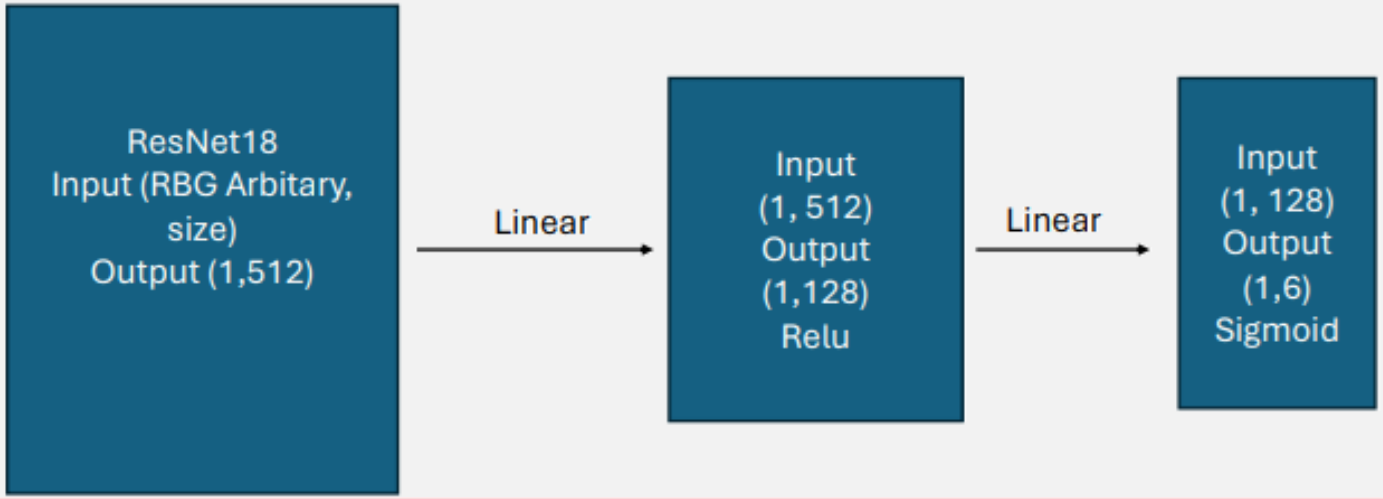
Convolutional neural network

- Why use a CNN?
  - Spectrogram is an image
  - Usage of vertical kernel
- Model trained on 2560 samples with 3 instruments
- Very poor performance
  - approx. 60% accuracy



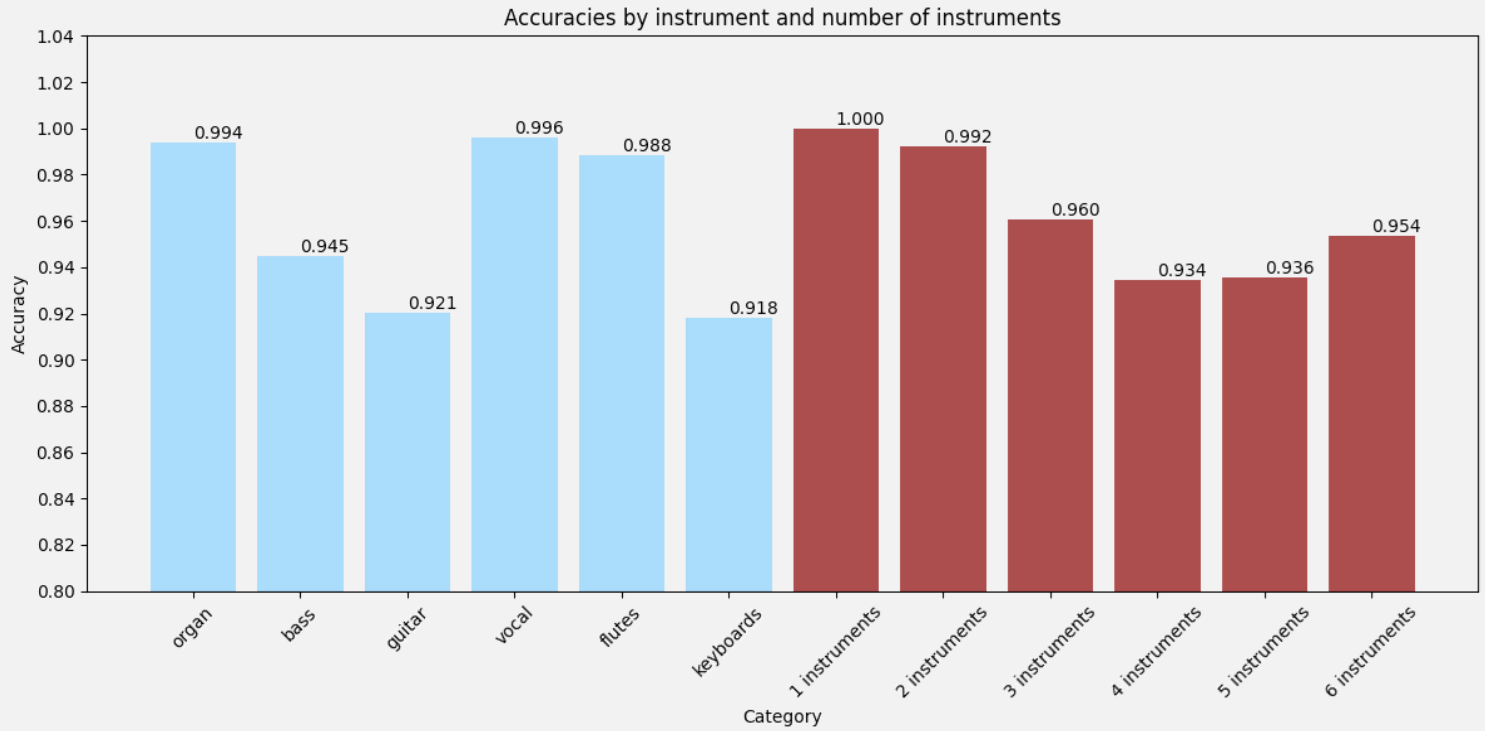
CNN with pretrained ResNet

- Idea: Why not just use somebody else work.
- Pretrained models for classification on hundreds of classes of color pictures.
- Resnet18 has convulutional layers, for details see appendix.
- Add a dense layer, froze the weights of the base model.



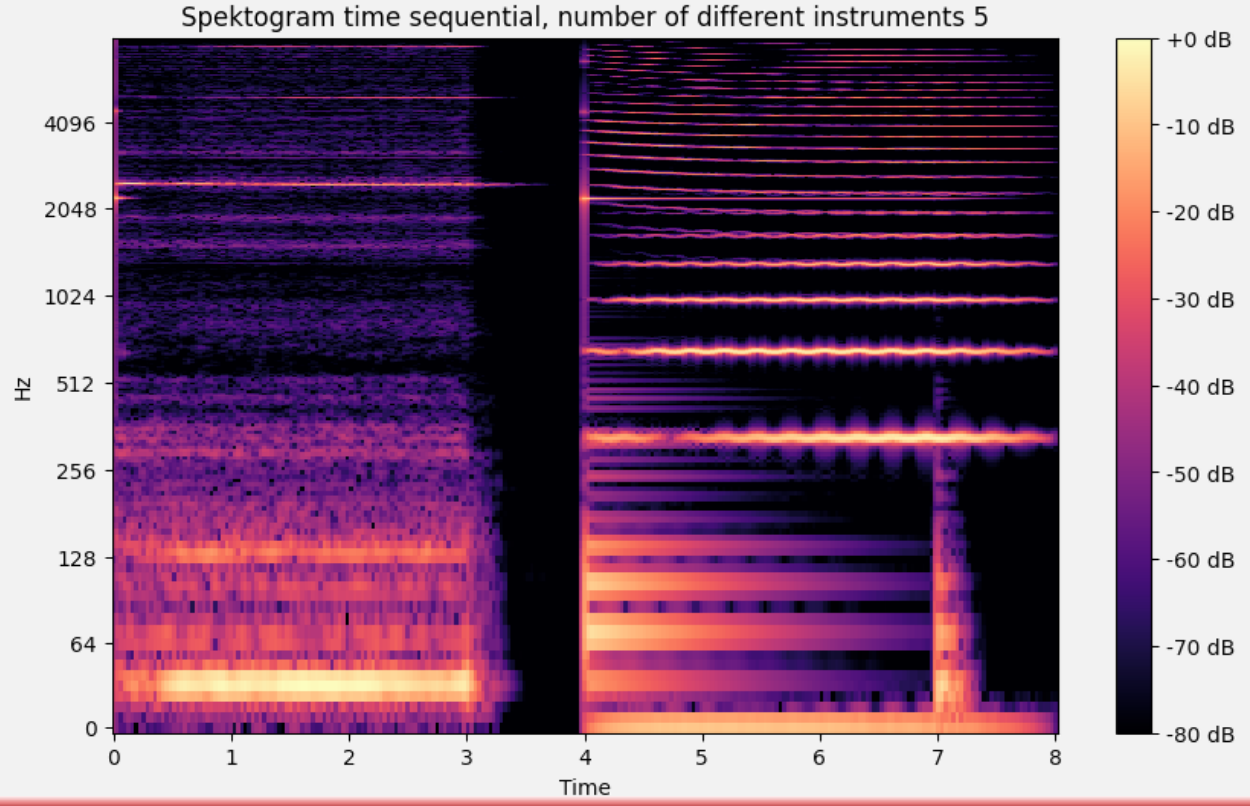
Results for mixed number of instruments

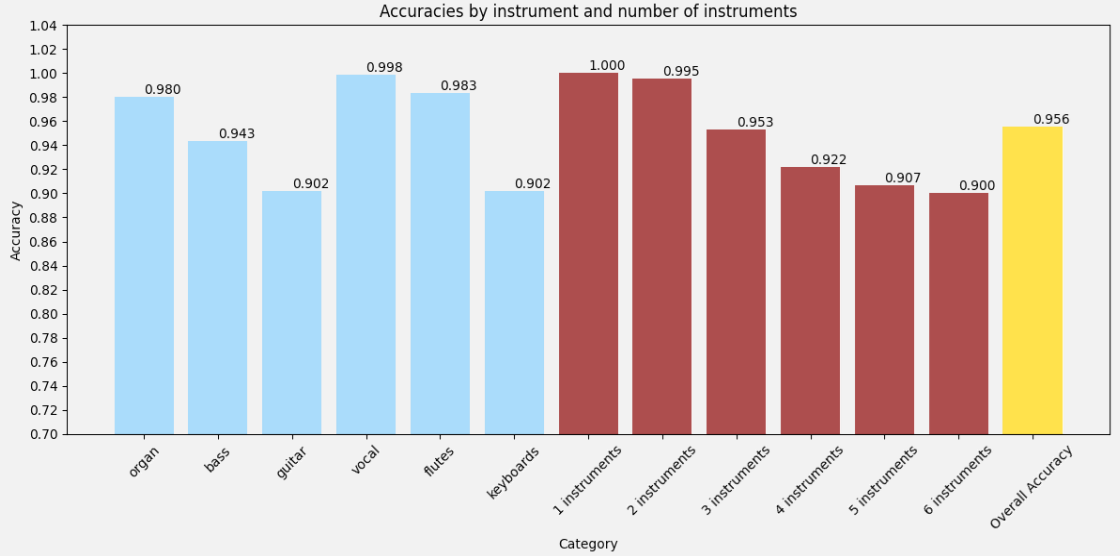
- Number of instruments varying from 1 – 6, 7800 total files
- Overall accuracy 96%, 13 epochs



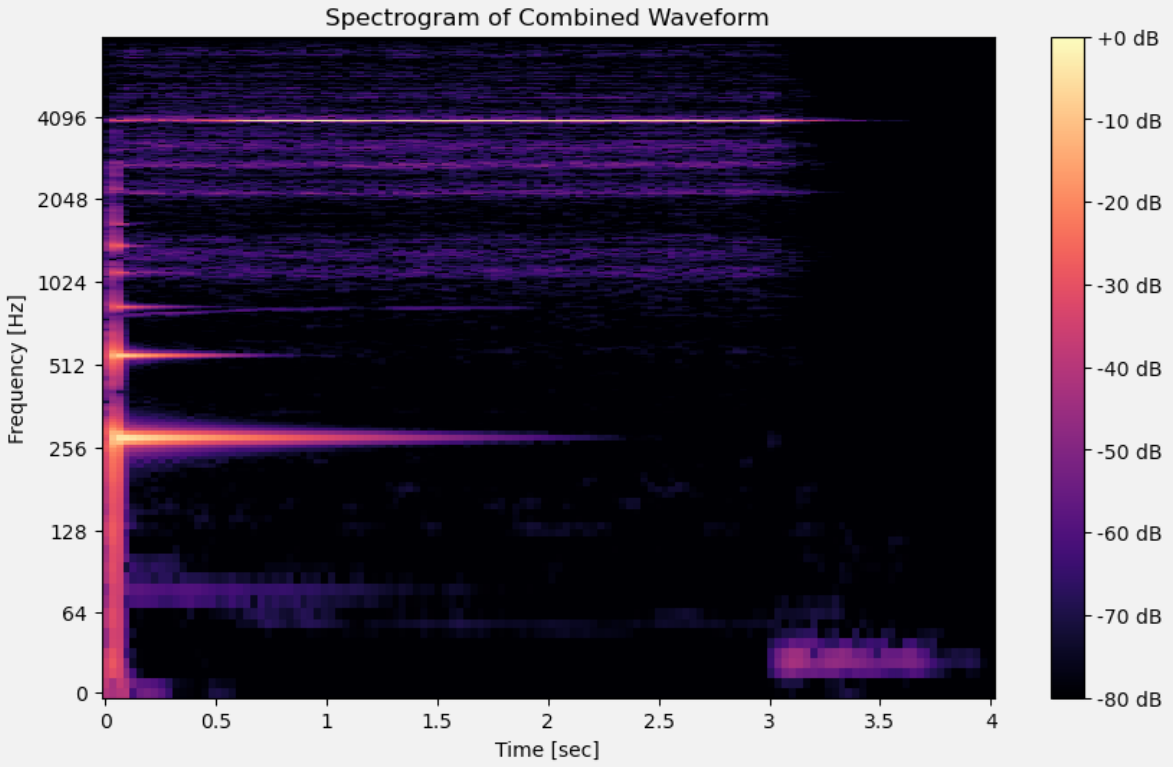
Results for mixed number of instruments and time scales

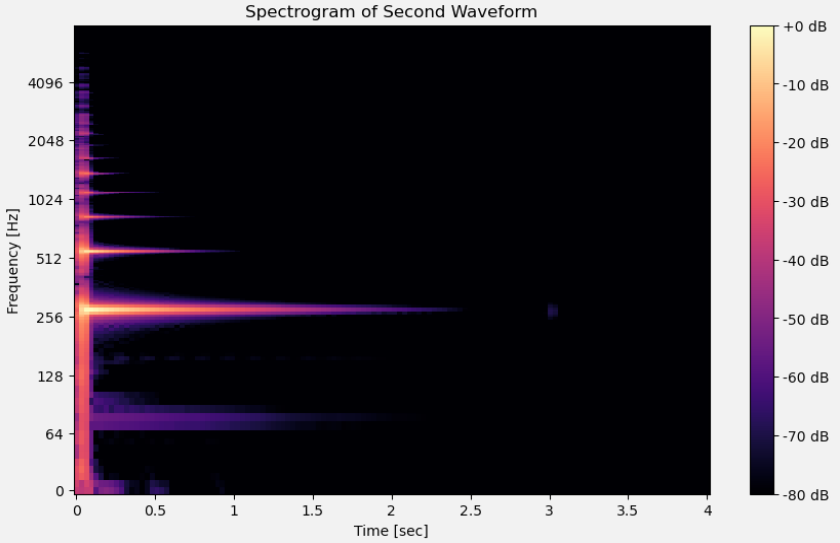
- The next idea: Music instruments changes with time in real music.
- 2 sound files with 3 random instruments, total of 5 instruments. 6000 datasets

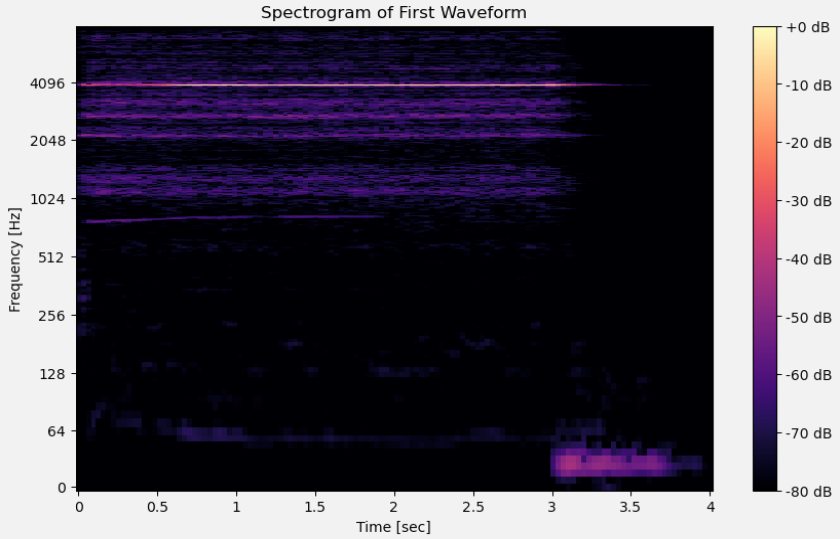




Audio Source Seperation



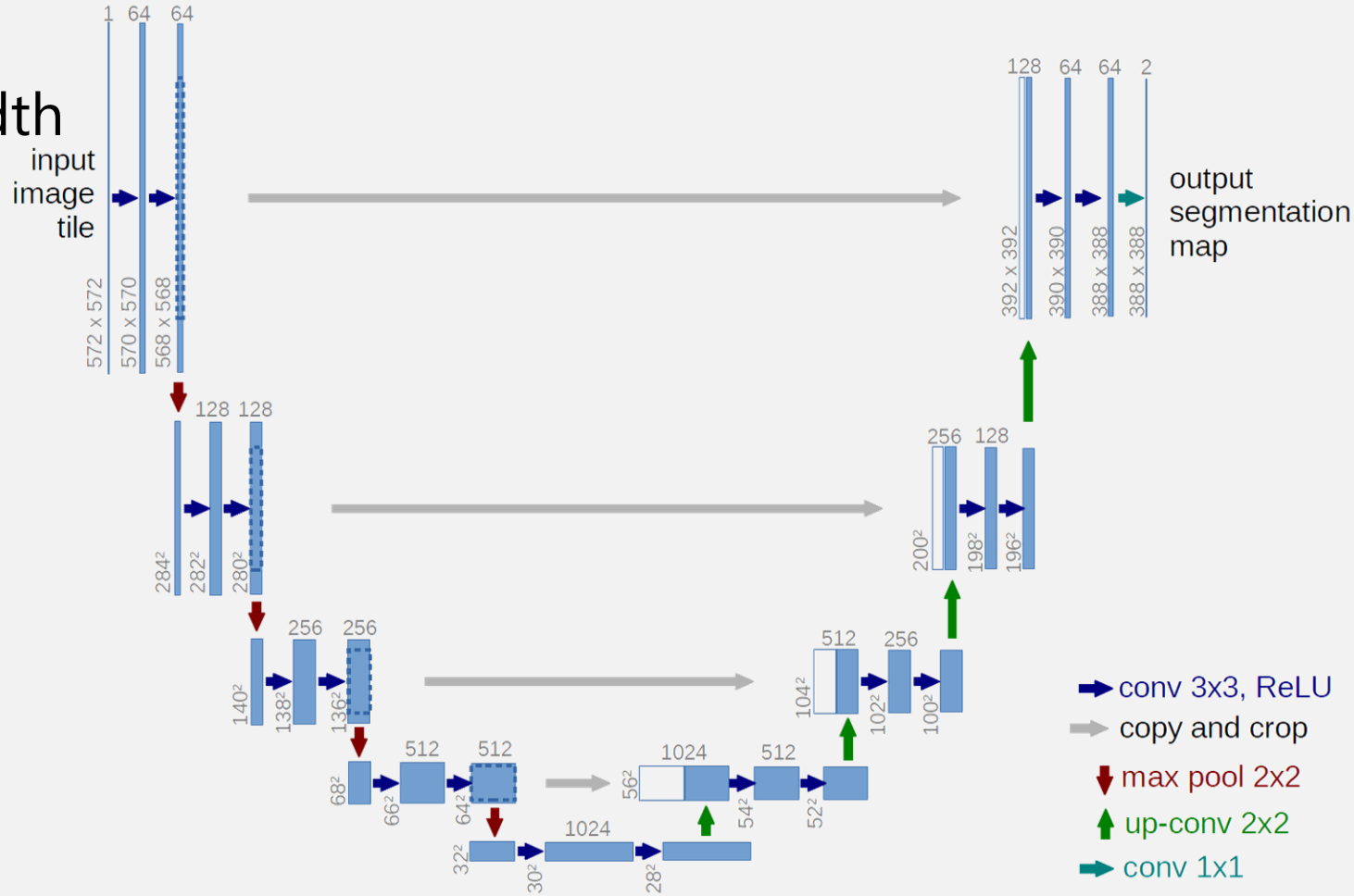




UNET: The CNN for image segmentation

Input and output

Batchsize, Channel, height, width

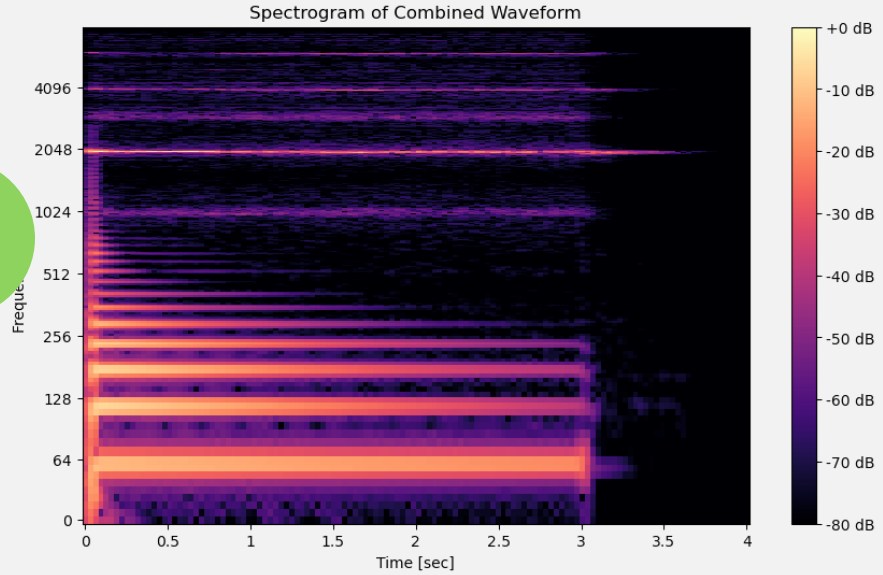


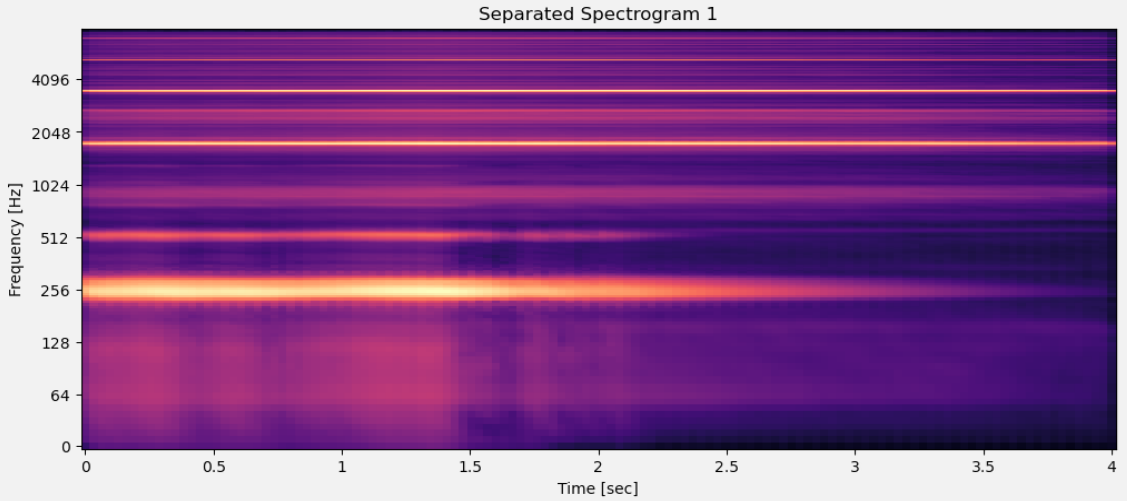
Simple Illustration of UNET

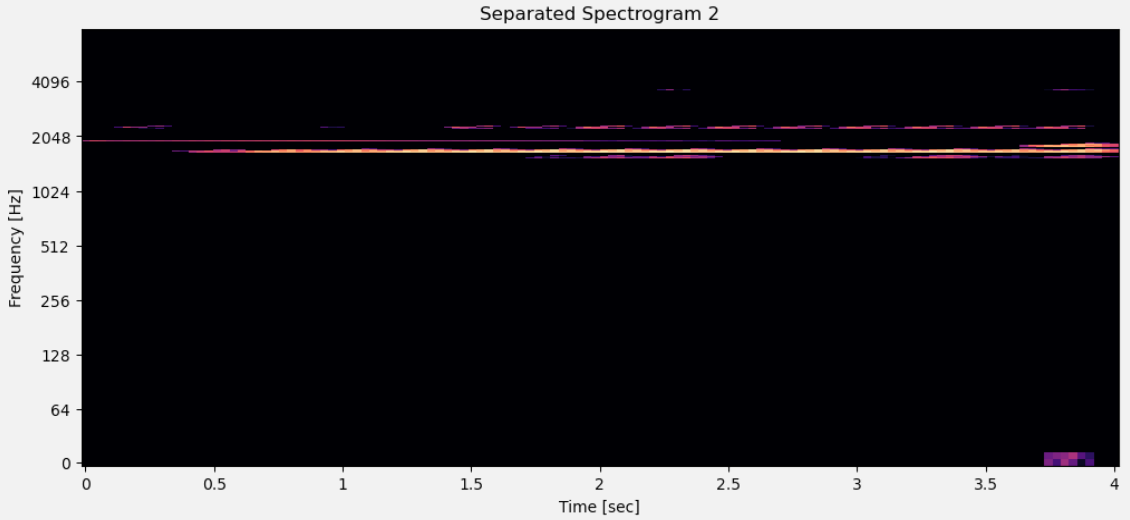
Source: https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/

Results with MSE Loss function

$$\text{MSE} = \frac{\sum_{i=1}^{N}(\hat{y}_i - yi)^2}{N}$$



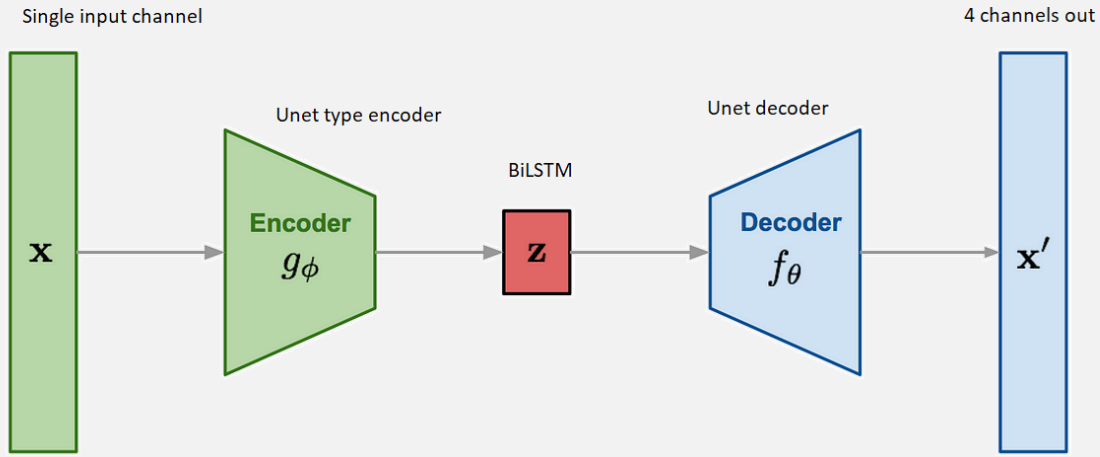




$$\text{SDR} = 10\log_{10}\left(\frac{\sum_t y(t)^2}{\sum_t (y(t) - \hat{y}(t))^2}\right)$$

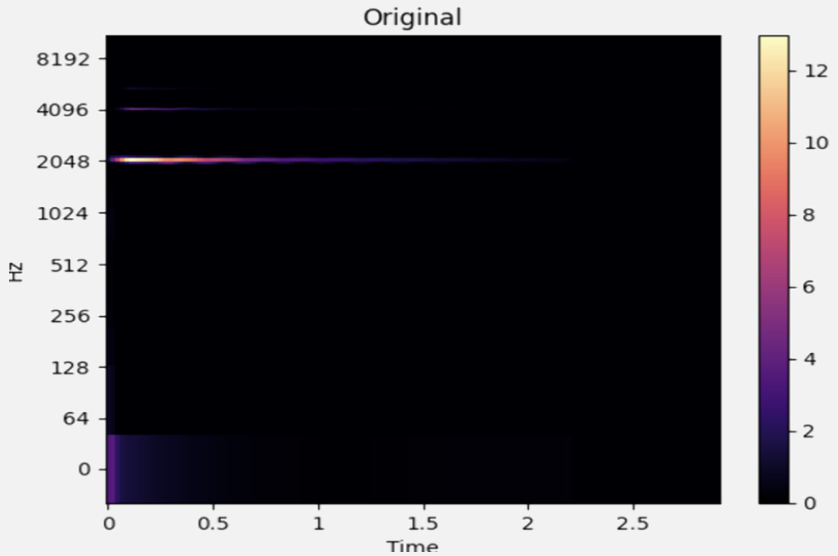
Unet with a central BiLSTM

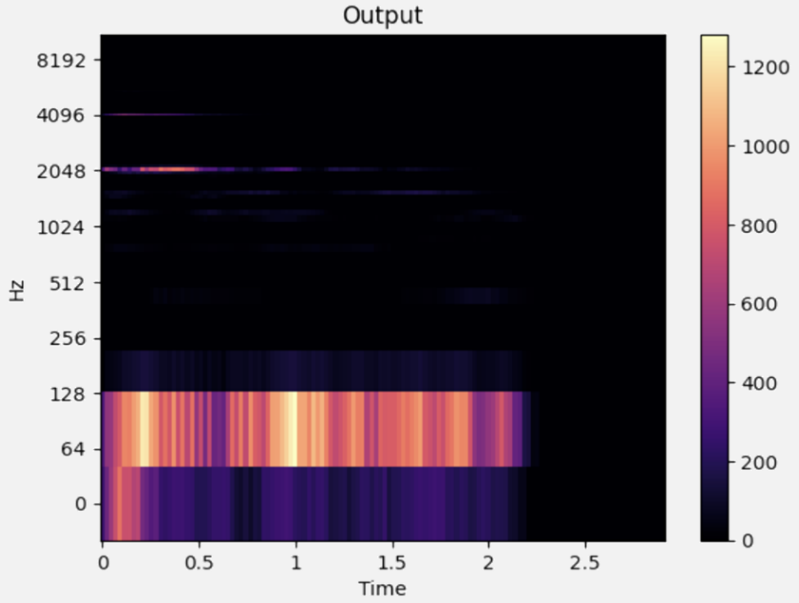
BiLSTM's (Bi-directional long short term memory), are able to learn how features should change in a (time-) series. And are often used for image segmentation tasks such as this one.

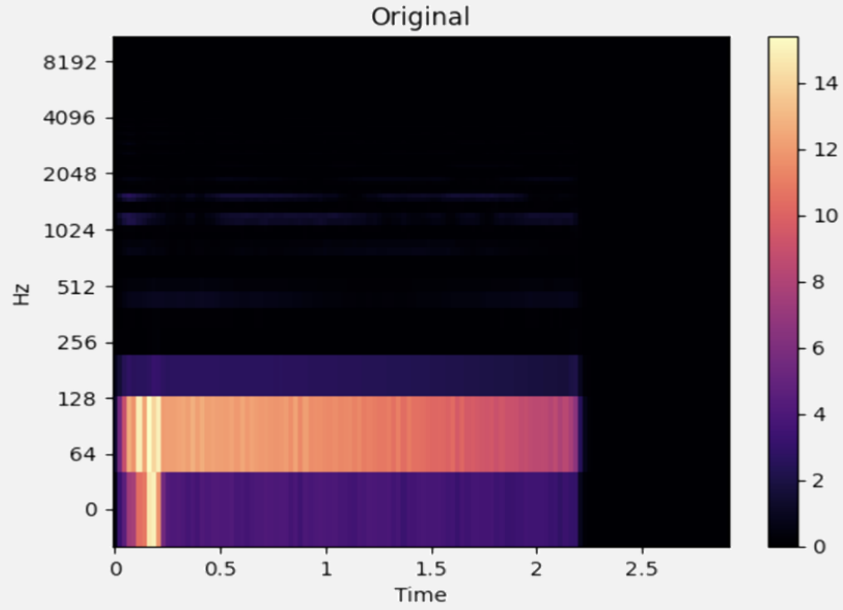


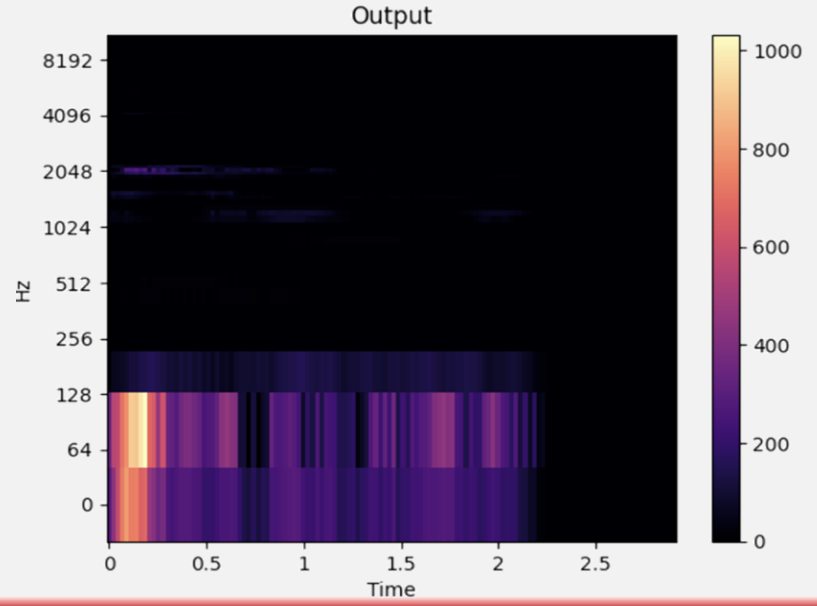
Preventing undesired local minimums

- Ideally, we would use a data set consisting of at least a 100.000 images, but the memory requirement for this is enormous. So we're forced to use a smaller one.
- However, it becomes harder to avoid local minimums, such as the model learning to spit out only zero spectrograms.
- In order to get the model out of these minimums, a series of penalties were introduced into the loss.
- Loss = -SDR + masked mse error + extra penalty for negative SDR scores.


Original
8192
4096
2048
1024
512
256
128
64
0
Hz
0
0.5
1
1.5
2
2.5
Time
12
10
8
6
4
2
0

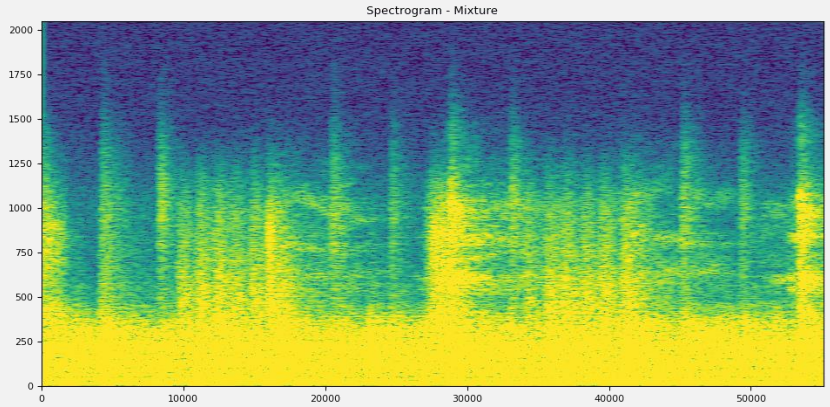

Output
8192
4096
2048
1024
512
256
128
64
0
Hz
0
0.5
1
1.5
2
2.5
Time
1200
1000
800
600
400
200
0


Original
8192
4096
2048
1024
512
256
128
64
0
Hz
0
0.5
1
1.5
2
2.5
Time
14
12
10
8
6
4
2
0

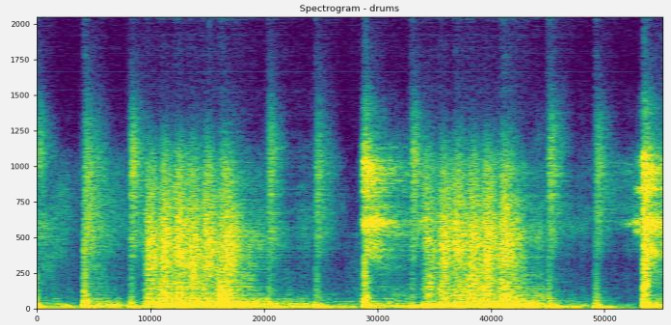

Output
8192
4096
2048
1024
512
256
128
64
0
Hz
0
0.5
1
1.5
2
2.5
Time
1000
800
600
400
200
0

Hybrid Demucs

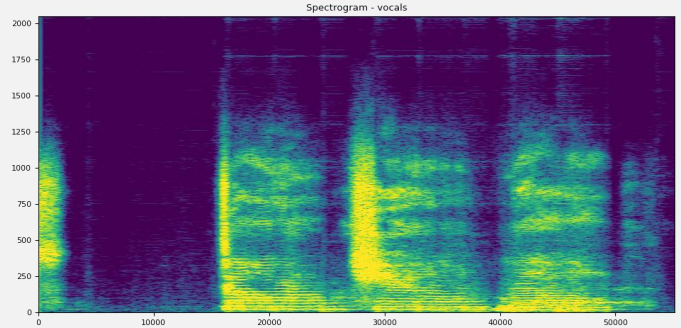
- Source separation using a hybrid of spectrograms and waveforms
- Winner of 2021 Sony music demixing challenge



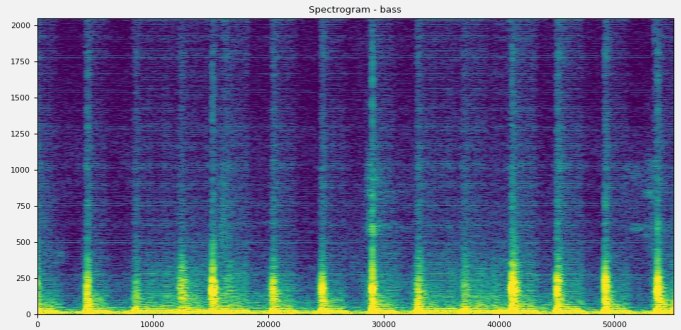
Mixed input



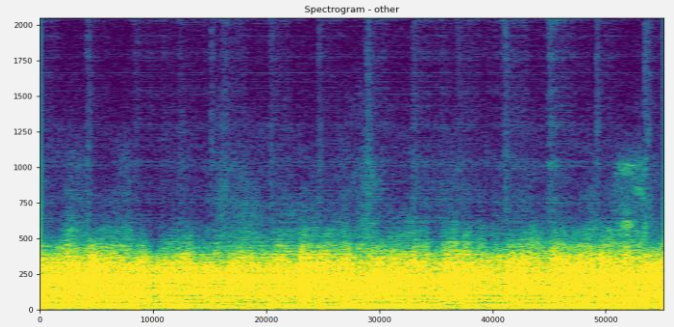
Drums



Vocals



Bass

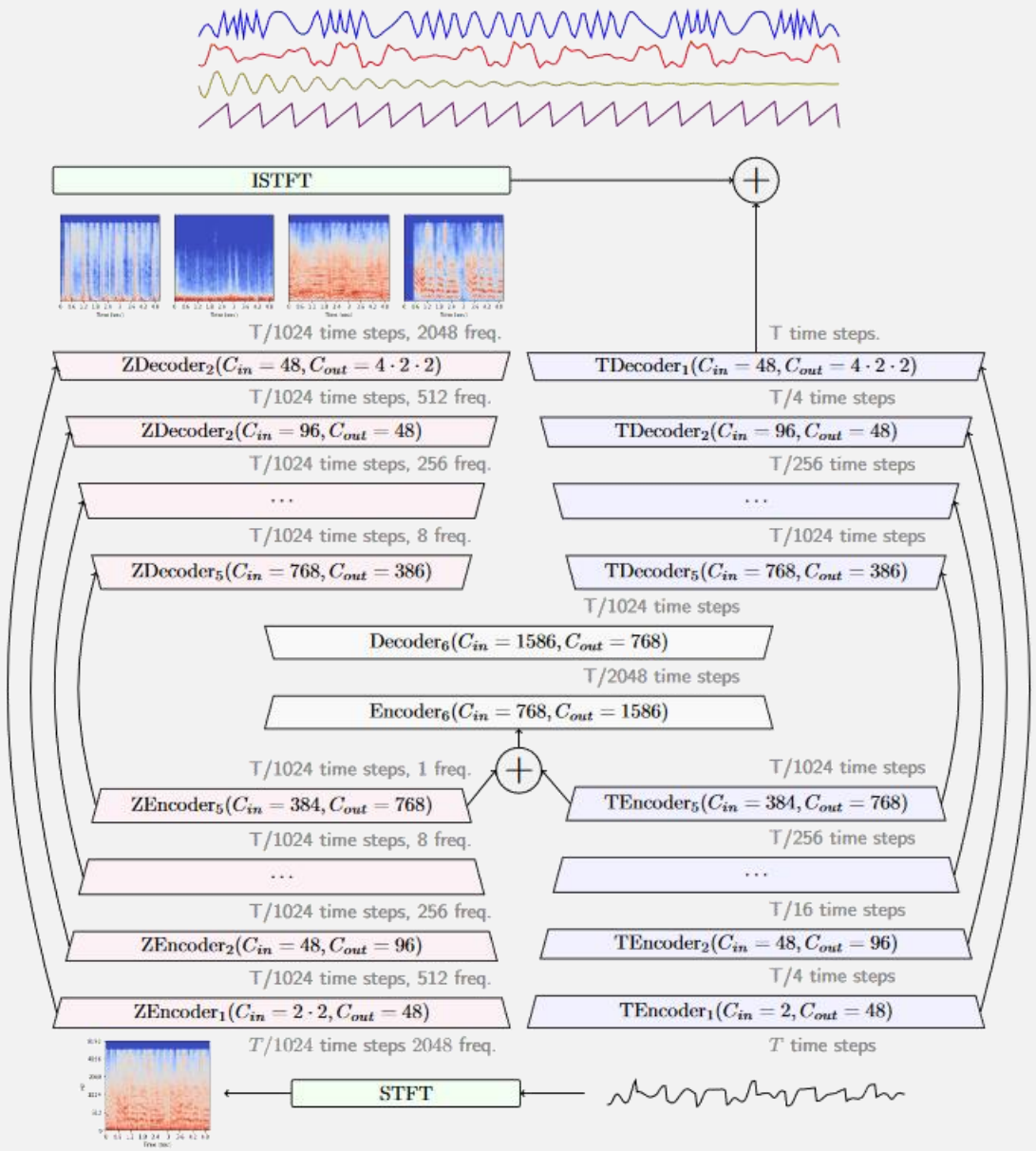


Residuals

https://doi.org/10.48550/arXiv.2111.03600

Demucs architecture

- U-net encoder/decoder architecture
- Very complex model
  - Uses separate temporal and spectral branches
  - Shared decoder layers



Applying Demucs to our data

Mixed input
(vocal, bass and guitar)

Bass

Drums

Residuals

Vocal

Summary

1) Single instrument classification:
   - 98% accuracy with BDT using XGBoost

2) Multi instrument classification most successful:
   - CNN with ResNet18 - accuracy of approx. 96 % (1-6 instruments)

3) Audio seperation is difficult...
   - Unet approach seems promising
   - We are computationally limited

Appendix

All code and trained models to be found at the GitHub page of the project:

https://github.com/JanMcKenzie/AudioNNRep

A. Model Specifications

1.1 xgboost_model.pkl
1.2 xgboost_model_new_input.pkl
2.1 XGBoost_multi_int
2.2 MLP_multi_inst_class.pth
2.3 MLP_multi_inst_class_V2_83.pth (too large for Github)
2.4 MLP_multi_inst_class_V2_85.pth (too large for Github)
2.5 MLP_multi_inst_class_V2_5_layer.pth (too large for Github)
2.6 CNN_pytorch_model.pth
2.7 resnet18_3_instruments.pth
2.8 resnet18_mix.pth
3.1 UNET_SourceSeperation_Simple.pth
3.2 UNET_SourceSeperation_Simple_SDR.pth
3.3 UNET_SourceSeperation_Bottleneck.pth
3.4 UNET_sourceseperation_BiLSTM

1.1 xgboost_model.pkl

Goal: Classification of instrument type used for input of single instrument
Training file: Classification_XGBoost_Luc.ipynb
Library: XGBClassifier
Data: http://download.magenta.tensorflow.org/datasets/nsynth/nsynth-valid.jsonwav.tar.gz
Data pre-processing: .wav to spectrogram
Training set: 4000 unique samples in classes 'keyboard', 'bass', 'flute' and 'guitar'
Validation set: 500 unique samples in classes 'keyboard', 'bass', 'flute' and 'guitar'
Testing set: 500 unique samples in classes 'keyboard', 'bass', 'flute' and 'guitar'
Model input shape: 1x36765 numpy array
Model output shape: single integer between [0,3]
Hyperparameters: objective='multi:softmax', num_class=4, n_estimators=100, max_depth=3, learning_rate=0.1, early_stopping_rounds=10
Performance: 98.4% accuracy on testing set

1.2 xgboost_model_new_input.pkl

Goal: Classification of instrument type used for input of single instrument
Training file: Classification_XGBoost_Luc_new_input.ipynb
Library: XGBClassifier
Data: http://download.magenta.tensorflow.org/datasets/nsynth/nsynth-valid.jsonwav.tar.gz
Data pre-processing: .wav to spectrogram
Training set: 1000 unique samples in classes 'keyboard', 'bass', 'flute' and 'guitar'
Validation set: 500 unique samples in classes 'keyboard', 'bass', 'flute' and 'guitar'
Testing set: 500 unique samples in classes 'keyboard', 'bass', 'flute' and 'guitar'
Model input shape: 1x129150 numpy array
Model output shape: single integer between [0,3]
Hyperparameters: objective='multi:softmax', num_class=4, n_estimators=100, max_depth=3, learning_rate=0.1, early_stopping_rounds=10
Performance: 97.8% accuracy on testing set

2.1 XGBoost_multi_int

Goal: Classification of presence/absence of 6 instruments in input file with 3 instruments
Training file: XGBoost_multi_class_luc.ipynb
Library: XGBClassifier
Data: http://download.magenta.tensorflow.org/datasets/nsynth/nsynth-valid.jsonwav.tar.gz
Data pre-processing: combination of 3 .wav files of different instruments into a single spectrogram
Training set: 4000 unique samples in of combined classes 'organ', 'bass', 'guitar', 'vocal', 'flutes' and 'keyboard'
Validation set: 500 unique samples in of combined classes 'organ', 'bass', 'guitar', 'vocal', 'flutes' and 'keyboard'
Testing set: 500 unique samples in of combined classes 'organ', 'bass', 'guitar', 'vocal', 'flutes' and 'keyboard'
Model type: individual model for each of the instruments
Model input shape: 1x129150 numpy array
Model output shape: single Boolean value
Hyperparameters: objective='binary:logistic', num_class=4, n_estimators=100, max_depth=3, learning_rate=0.1, early_stopping_rounds=10 (for each model)
Performance: total accuracy of 74.7%. Individual accuracies of: organ=74%, bass=62%, guitar=67%, vocal=96%, flutes=84%, keyboard=65%

2.2 MLP_multi_inst_class.pth

Goal: Classification of presence/absence of 6 instruments in input file with 3 instruments
Training file: MLP_multi_class_luc.ipynb
Library: PyTorch
Data: http://download.magenta.tensorflow.org/datasets/nsynth/nsynth-valid.jsonwav.tar.gz
Data pre-processing: combination of 3 .wav files of different instruments into a single spectrogram, min-max normalized
Training set: 4000 unique samples in of combined classes 'organ', 'bass', 'guitar', 'vocal', 'flutes' and 'keyboard'
Validation set: 500 unique samples in of combined classes 'organ', 'bass', 'guitar', 'vocal', 'flutes' and 'keyboard'
Testing set: 500 unique samples in of combined classes 'organ', 'bass', 'guitar', 'vocal', 'flutes' and 'keyboard'
Model architecture: 3 hidden layers (neurons: input, 64, 32, output). ReLu for each layer.
Model input shape: 1x129150 torch tensor
Model output shape: 1x6 torch tensor with floating point numbers
Hyperparameters: optimizer=Adam, learning_rate=0.001, loss_function = torch.sum((y_pred – y_true)**2), epochs=300, early_stopping_patience=5
Performance: Accuracy of 82.9%

2.3 MLP_multi_inst_class_V2_83.pth

Goal: Classification of presence/absence of 6 instruments in input file with variable number of instruments (from 1 to 6)
Training file: MLP_multi_class_luc.ipynb
Library: PyTorch
Data: http://download.magenta.tensorflow.org/datasets/nsynth/nsynth-valid.jsonwav.tar.gz
Data pre-processing: combination of 1-6 .wav files of different instruments into a single spectrogram, min-max normalized
Training set: 4800 unique samples in of combined classes 'organ', 'bass', 'guitar', 'vocal', 'flutes' and 'keyboard'
Validation set: 600 unique samples in of combined classes 'organ', 'bass', 'guitar', 'vocal', 'flutes' and 'keyboard'
Testing set: 600 unique samples in of combined classes 'organ', 'bass', 'guitar', 'vocal', 'flutes' and 'keyboard'
Model architecture: 3 hidden layers (neurons: input, 64, 32, output). ReLu for each layer.
Model input shape: 1x129150 torch tensor
Model output shape: 1x6 torch tensor with floating point numbers
Hyperparameters: optimizer=Adam, learning_rate=0.001, loss_function = torch.sum((y_pred – y_true)**2), epochs=300, early_stopping_patience=5
Performance: Accuracy of 83.1% (got stuck within 300 epochs)

2.4 MLP_multi_inst_class_V2_85.pth

Goal: Classification of presence/absence of 6 instruments in input file with variable number of instruments (from 1 to 6)
Training file: MLP_multi_class_luc.ipynb
Library: PyTorch
Data: http://download.magenta.tensorflow.org/datasets/nsynth/nsynth-valid.jsonwav.tar.gz
Data pre-processing: combination of 1-6 .wav files of different instruments into a single spectrogram, min-max normalized
Training set: 4800 unique samples in of combined classes 'organ', 'bass', 'guitar', 'vocal', 'flutes' and 'keyboard'
Validation set: 600 unique samples in of combined classes 'organ', 'bass', 'guitar', 'vocal', 'flutes' and 'keyboard'
Testing set: 600 unique samples in of combined classes 'organ', 'bass', 'guitar', 'vocal', 'flutes' and 'keyboard'
Model architecture: 3 hidden layers (neurons: input, 64, 32, output). ReLu for each layer.
Model input shape: 1x129150 torch tensor
Model output shape: 1x6 torch tensor with floating point numbers
Hyperparameters: optimizer=Adam, learning_rate=0.001, loss_function = torch.sum((y_pred – y_true)**2), epochs=300, early_stopping_patience=20
Performance: Accuracy of (85.16 ± 0.15)%

2.5 MLP_multi_inst_class_V2_5_layer.pth

Goal: Classification of presence/absence of 6 instruments in input file with variable number of instruments (from 1 to 6)
Training file: MLP_multi_class_luc.ipynb
Library: PyTorch
Data: http://download.magenta.tensorflow.org/datasets/nsynth/nsynth-valid.jsonwav.tar.gz
Data pre-processing: combination of 1-6 .wav files of different instruments into a single spectrogram, min-max normalized
Training set: 4800 unique samples in of combined classes 'organ', 'bass', 'guitar', 'vocal', 'flutes' and 'keyboard'
Validation set: 600 unique samples in of combined classes 'organ', 'bass', 'guitar', 'vocal', 'flutes' and 'keyboard'
Testing set: 600 unique samples in of combined classes 'organ', 'bass', 'guitar', 'vocal', 'flutes' and 'keyboard'
Model architecture: 5 hidden layers (neurons: input, 256, 128, 64, 32, output). ReLu for each layer.
Model input shape: 1x129150 torch tensor
Model output shape: 1x6 torch tensor with floating point numbers
Hyperparameters: optimizer=Adam, learning_rate=0.001, loss_function = torch.sum((y_pred – y_true)**2), epochs=300, early_stopping_patience=20
Performance: Accuracy of 80.8%

2.6 CNN_pytorch_model.pth

Goal: Classification of presence/absence of 6 instruments in input file with variable number of
Training file: CNN_pytorch_model.ipynb
Library: PyTorch
Data: http://download.magenta.tensorflow.org/datasets/nsynth/nsynth-test.jsonwav.tar.gz
Data pre-processing: combination of 3 .wav files of different instruments into a single spectrogram, min-max normalized
Training set: 2560 unique samples in of combined classes 'organ', 'bass', 'guitar', 'vocal', 'flutes' and 'keyboard'
Validation set: 640 unique samples in of combined classes 'organ', 'bass', 'guitar', 'vocal', 'flutes' and 'keyboard'
Testing set: 640 unique samples in of combined classes 'organ', 'bass', 'guitar', 'vocal', 'flutes' and 'keyboard'

- **Model architecture and HP:** Conv layer: Conv2d(32, 64, kernel_size=(2, 126), padding=(1, 64)), ReLU
- Max pooling: MaxPool2d(kernel_size=(2, 2), stride=2), Conv layer: Conv2d(64, 32, kernel_size=(3, 3), padding=1), ReLU. Max pooling: MaxPool2d(kernel_size=(2, 2), stride=2), Dropout: Dropout(0.2)

Then fully connected layers: nn.Linear(8192, 512) ReLu, Linear(512, 32) ReLu, Linear(32, 6) Sigmoid.
Model input shape: `[32, 1025, 126] (32 is batch size, 1025, 126 is the spectrogram)`
Model output shape: `[32, 6] (32 is batch size, 6 is prob of presence of each instr.)`
Performance: 61.2 % accuracy
Loss function: sum(abs(Y_{pred} -Y_{True}))

2.7 resnet18_3_instruments.pth

Goal: Classification of instruments present from audiofiles.
Training file: `Kopi_af_imagenet_copilot.ipynb`
Library: PyTorch
Data: 4 second audio files, converted to spectrograms
Data pre-processing: Conversion of spectrograms to RGB images.
Training set: 780 unique samples of 3 instrument mixed spectrograms combined classes 'organ', 'bass', 'guitar', 'vocal', 'flutes' and 'keyboard'.
Validation set: 780 unique samples of 3 instrument mixed spectrograms combined classes 'organ', 'bass', 'guitar', 'vocal', 'flutes' and 'keyboard'.
Testing set: 6240 unique samples of 3 instrument mixed spectrograms combined classes 'organ', 'bass', 'guitar', 'vocal', 'flutes' and 'keyboard'.
Model architecture: Pretrained ResNet18 with added layers:

- Adaptive average pooling layer to (1, 1)
- Fully connected layer: Linear(512, 128) with ReLU and Dropout
- Output layer: Linear(128, 6) with Sigmoid

Model input shape: (3, 1025, 126) for 1 sound file, Model reshapes to (3, 256, 256), and crops to (3, 224,224)
Model output shape: 6-dimensional output.
Hyperparameters:

- Optimizer: Adam, Learning rate: 1e-4, Epochs: 100, Early stopping patience: 10, loss function = abs(torch.sum((y_pred – y_true)))/N

2.8 resnet18_mix.pth

Goal: Classification of instruments present from audiofiles.
Training file: `Kopi_af_imagenet_copilot.ipynb`
Library: PyTorch
Data: 4 second audio files, converted to spectrograms
Data pre-processing: Conversion of spectrograms to RGB images.
Training set: 780 unique samples of mixed spectrograms combined classes 'organ', 'bass', 'guitar', 'vocal', 'flutes' and 'keyboard'.
Validation set: 780 unique samples of mixed spectrograms combined classes 'organ', 'bass', 'guitar', 'vocal', 'flutes' and 'keyboard'.
Testing set: 6240 unique samples of mixed spectrograms combined classes 'organ', 'bass', 'guitar', 'vocal', 'flutes' and 'keyboard'.
Model architecture: Pretrained ResNet18 with added layers:

- Adaptive average pooling layer to (1, 1)
- Fully connected layer: Linear(512, 128) with ReLU and Dropout
- Output layer: Linear(128, 6) with Sigmoid

Model input shape: (3, 1025, 126) for 1 sound file, (3,1025,251) for 2 soundfiles. Model reshapes to (3, 256, 256), and crops to (3, 224,224)
Model output shape: 6-dimensional output.
Hyperparameters:

- Optimizer: Adam, Learning rate: 1e-4, Epochs: 100 (usually not needed), Early stopping patience: 10, loss_function = abs(torch.sum((y_pred – y_true)))/N

Accuracy: 96.1%

3. 1 UNET_SourceSeperation_Simple.pth

Goal: Seperation of two instruments, Electronic bass and Acoustic Flute.
Training file: UNET_EB_AF.ipynb
Library: PyTorch
Data: http://download.magenta.tensorflow.org/datasets/nsynth/nsynth-valid.jsonwav.tar.gz
Data pre-processing: combination of 1-2 .wav files of different instruments into a single spectrogram.
Training set: 4200 unique samples of combined classes 'Electronic bass' and 'Acoustic Flute'.
Validation set: 2 * Spectrogram of .wav files used for generating the training set.
Testing set: Validated on a single file generated according to the training set.
Model architecture: 7 downsampling and 7 upsampling layers. With skip connections between.
Model input shape: (Batch size, 1, width, height) = (60, 1, 1025, 126)
Model output shape: (Batch size, 1, width, height) = (60, 2, 1025, 126)
Loss function: MSELoss with a penalty term, Penalty to try and enforce no new features.
Hyperparameters: optimizer=SGD, learning_rate=0.01, epochs=300, early_s
topping_patience=5.

3. 2 UNET_SourceSeperation_Simple_SDR.pth

Goal: Seperation of two instruments, Electronic bass and Acoustic Flute.
Training file: UNET_EB_AF_simple.ipynb
Library: PyTorch
Data: http://download.magenta.tensorflow.org/datasets/nsynth/nsynth-valid.jsonwav.tar.gz
Data pre-processing: combination of 1-2 .wav files of different instruments into a single spectrogram.
Training set: 4200 unique samples of combined classes 'Electronic bass' and 'Acoustic Flute'.
Validation set: 2 * Spectrogram of .wav files used for generating the training set.
Testing set: Validated on a single file generated according to the training set.
Model architecture: 7 downsampling and 7 upsampling layers. With skip connections between.
Model input shape: (Batch size, 1, width, height) = (60, 1, 1025, 126)
Model output shape: (Batch size, 1, width, height) = (60, 2, 1025, 126)
Loss function: SDRLoss with a penalty term, Penalty to try and enforce no new features.
Hyperparameters: optimizer=ADAM, learning_rate=0.01, epochs=300, early_s topping_patience=5.

3.3 UNET_SourceSeperation_Bottleneck.pth

Goal: Seperation of two instruments, Electronic bass and Acoustic Flute.
Training file: UNET_EB_AF.ipynb
Library: PyTorch
Data: http://download.magenta.tensorflow.org/datasets/nsynth/nsynth-valid.jsonwav.tar.gz
Data pre-processing: combination of 1-2 .wav files of different instruments into a single spectrogram.
Training set: 4200 unique samples of combined classes 'Electronic bass' and 'Acoustic Flute'.
Validation set: 2 * Spectrogram of .wav files used for generating the training set.
Testing set: Validated on a single file generated according to the training set.
Model architecture: 7 downsampling and 7 upsampling layers. With skip connections between.
Model input shape: (Batch size, 1, width, height) = (60, 1, 1025, 126)
Model output shape: (Batch size, 1, width, height) = (60, 2, 1025, 126)
Loss function: MSELoss with a penalty term, Penalty to try and enforce no new features.
Hyperparameters: optimizer=ADAM, learning_rate=0.01, epochs=300, early_stopping_patience=5 .

3.4 UNET_sourceseperation_BiLSTM

Goal: Segmentation of multiple instruments
Training file: UnetBiLSTM.py, nu_gen_mel_spectro
Library: pytorch

Data: http://download.magenta.tensorflow.org/datasets/nsynth/nsynth-valid.jsonwav.tar.gz
Data pre-processing: combination of 1-4 .wav files of different instruments into a single spectrogram.
Training set: 1500 unique spectrograms of combined wav files.
Validation set: Cut spectrograms from the training set
Testing set: Individual testing
Model architecture: Downsampling path of 2 encoders, a central BiLSTM layer and a upsampling path of 2 decoders (C)
Model input shape: (2, 1, 128, 126)
Model output shape: (2, 2 - 4, 128, 126)
Hyperparameters: hidden_layers = 500, num_layers = 1

B. Elaboration on model architectures and data

1.1 ResNet18 model data generation

- RBG files were generated by adding two extra layers of 0 in the same size as the spectrograms

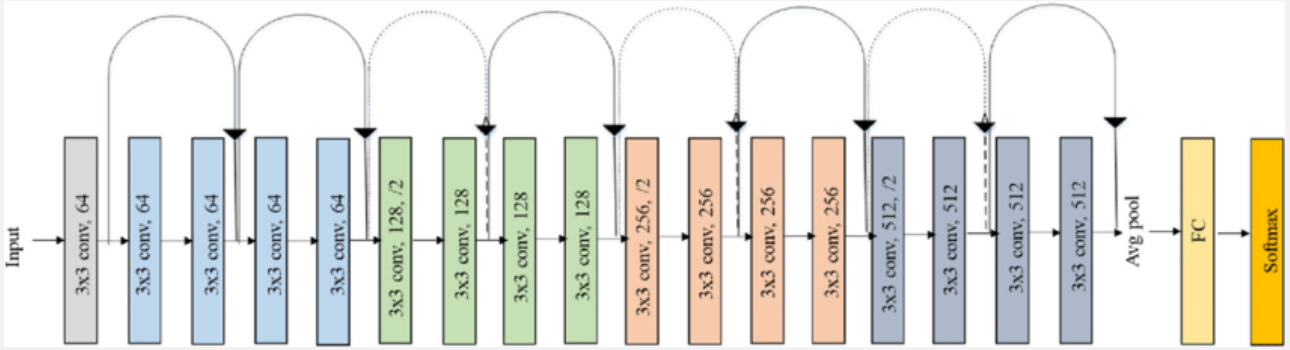
Methods tried to optimize data generation for 3 mixed instruments.

- Spectrograms more compact by changing size from (3,1025,126) to (3, 205, 126). Running code on 24000 samples. Decreased accuracy from 98% to 95% accuracy
- Save data and load in batches of 5000 files. Led to local storage problems
- Best method was training the data on 10000 files, after training use best trained model as an initial guess. Then train on 10000 new files. This showed slight increase in accuracy of 0.5%.

1.2 ResNet18 model structure and idea behind it

ResNet-18 is a convolutional network with 18 layers. The main idea behind it is using skip connections to avoid the vanishing gradient problem. The vanishing gradient problem occurs because the initial weights of the neural network are randomized, and when we have a deep neural network, it takes a long time during the initial training phase for backpropagation to adjust the weights effectively. This makes it difficult to adjust the weights in the initial layers during backpropagation.

The network therefor works in blocks with a skip connection between them. The skip connection work by saving the initial input to a block and concatenating or adding it in the output of the block.
Watch the following video for wonderful explanation.
https://www.youtube.com/watch?v=Q1JCrG1bJ-A&ab_channel=ProfessorBryce



ResNet18 figure, source:
A Deep Learning Approach for Automated Diagnosis and Multi-Class Classification of Alzheimer's Disease Stages Using Resting-State fMRI and Residual Neural Networks - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Original-ResNet-18-Architecture_fig1_336642248 [accessed 10 Jun, 2024]

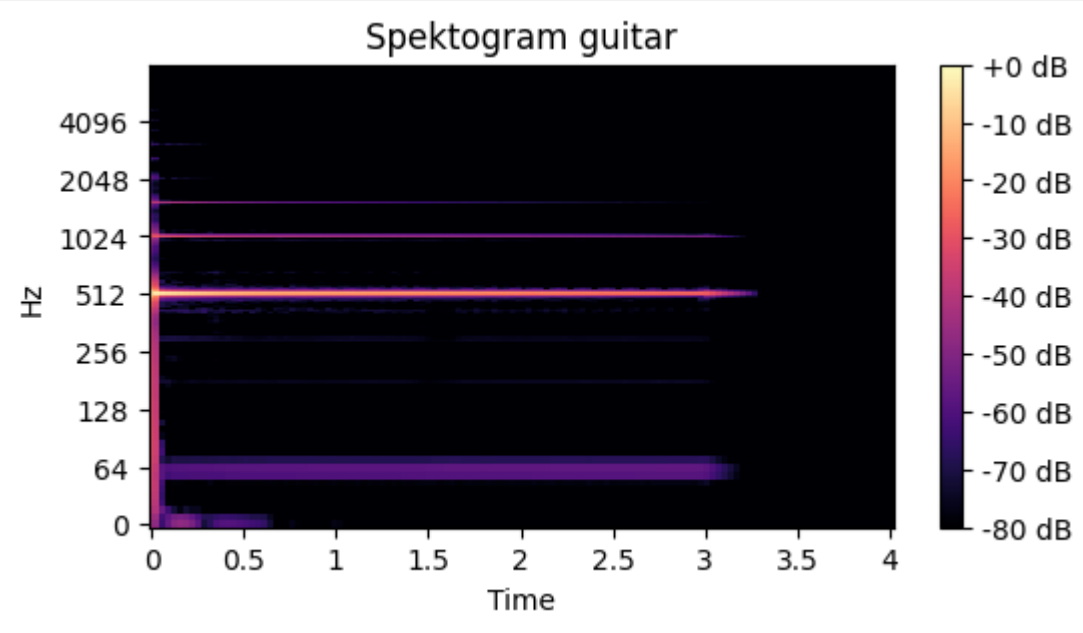
1.2 ResNet18 model structure and idea behind it

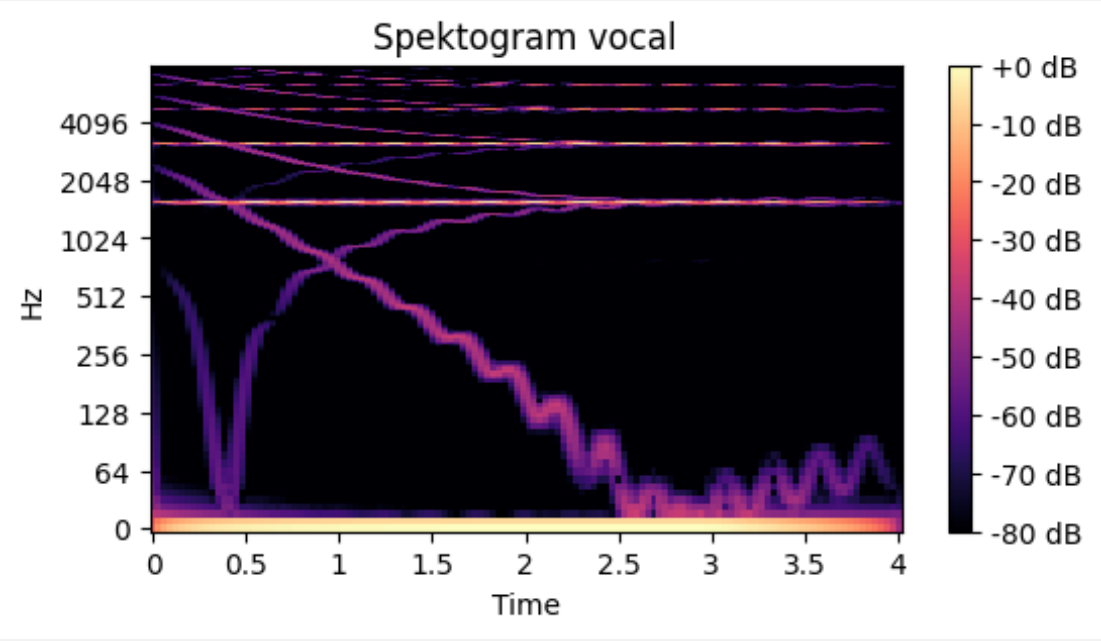
Base structure of the Resnet18 model.

Layer Name	Output Size	ResNet-18
conv1	$112 \times 112 \times 64$	7×7, 64, stride 2
conv2_x	$56 \times 56 \times 64$	3×3 max pool, stride 2
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$
conv3_x	$28 \times 28 \times 128$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$
conv4_x	$14 \times 14 \times 256$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$
conv5_x	$7 \times 7 \times 512$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$
average pool	$1 \times 1 \times 512$	7×7 average pool

1.3 ResNet18 why vocal is easier to recognize

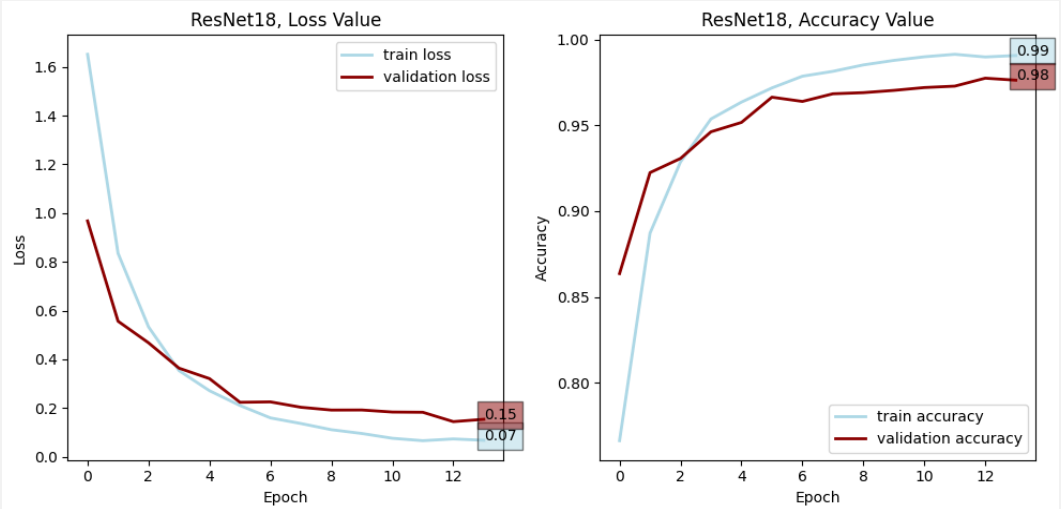
Spectrograms of different instruments.





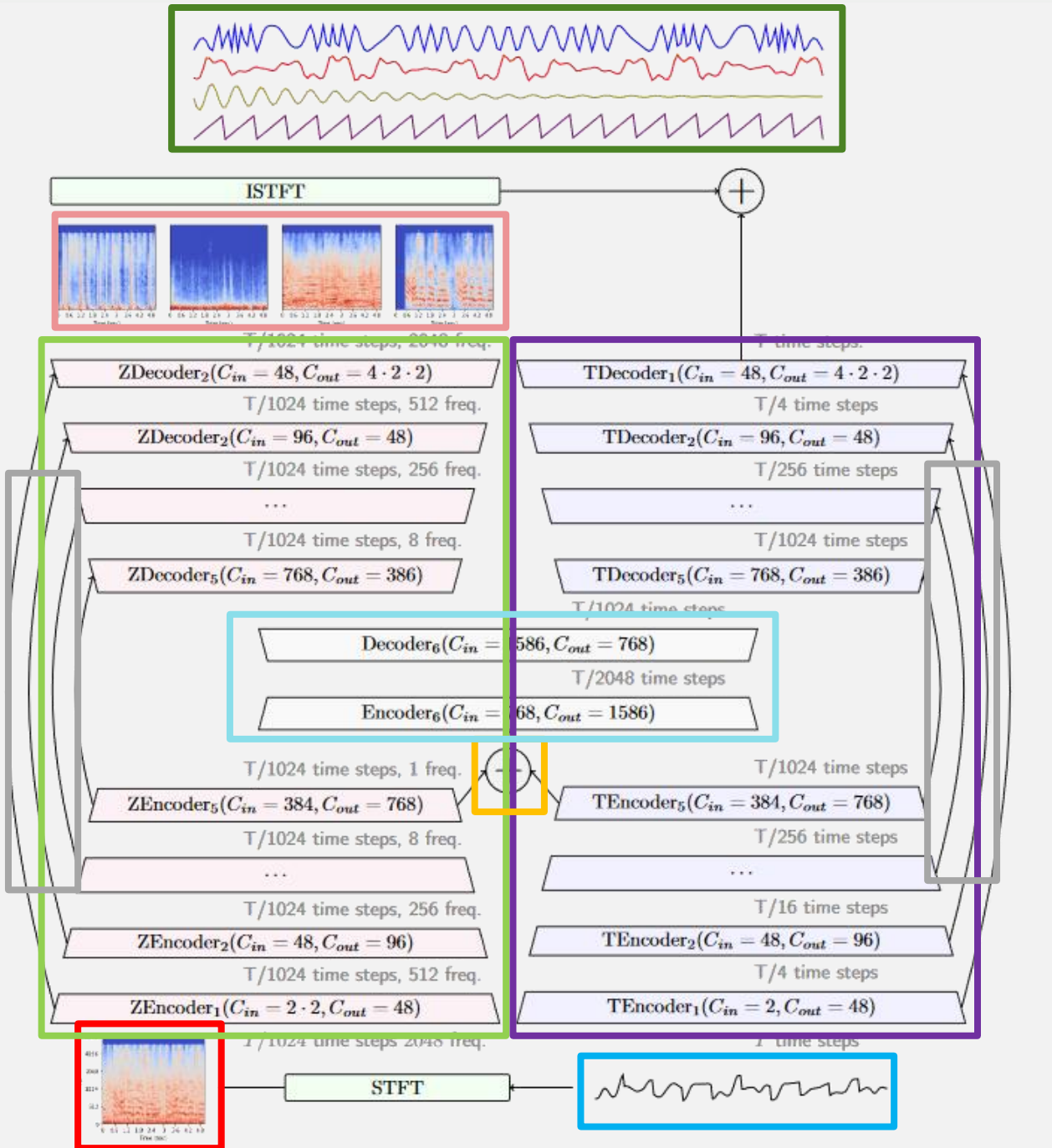
1.4 Results for mix of 3 instruments ResNet18 and ResNet50

- Memory problems handling large data sets.
  See problems and handling in appendix
- Each spectrogram consist of 3 out 6 random of instruments playing
- 10000 unique mixed files
- Similar results, ResNet 18, chosen for efficiency



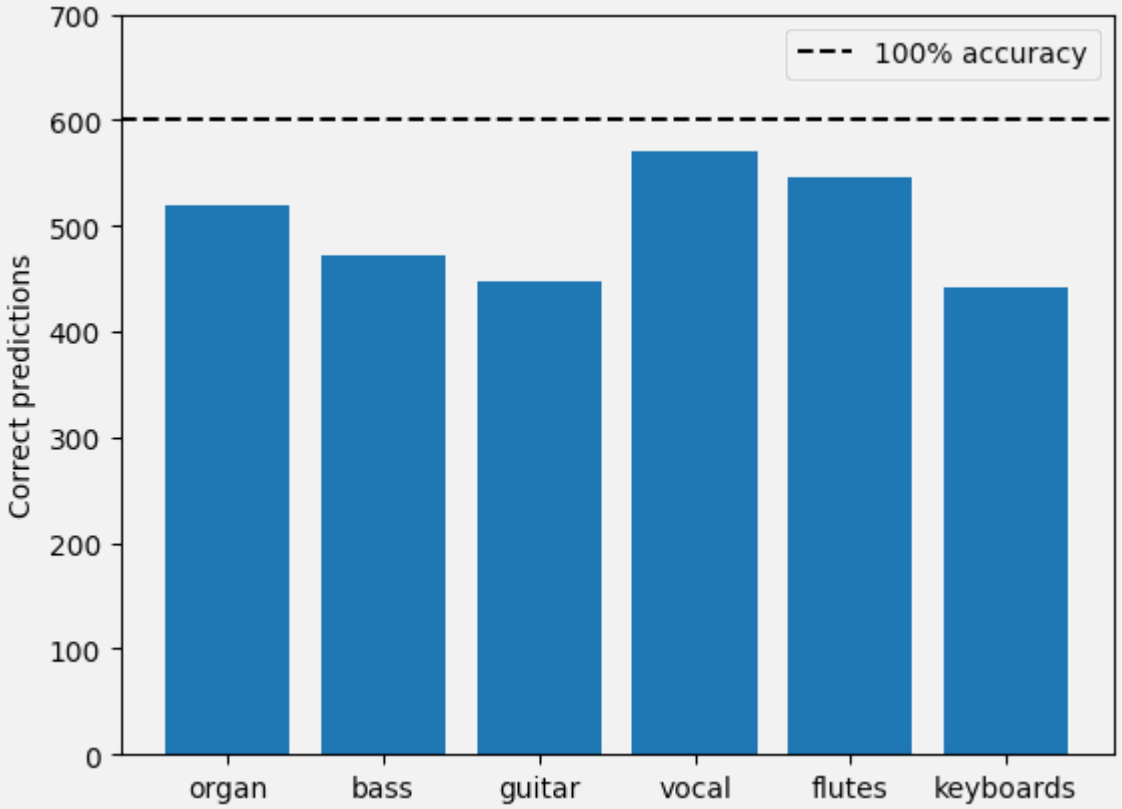
2.1 Hybrid demucs

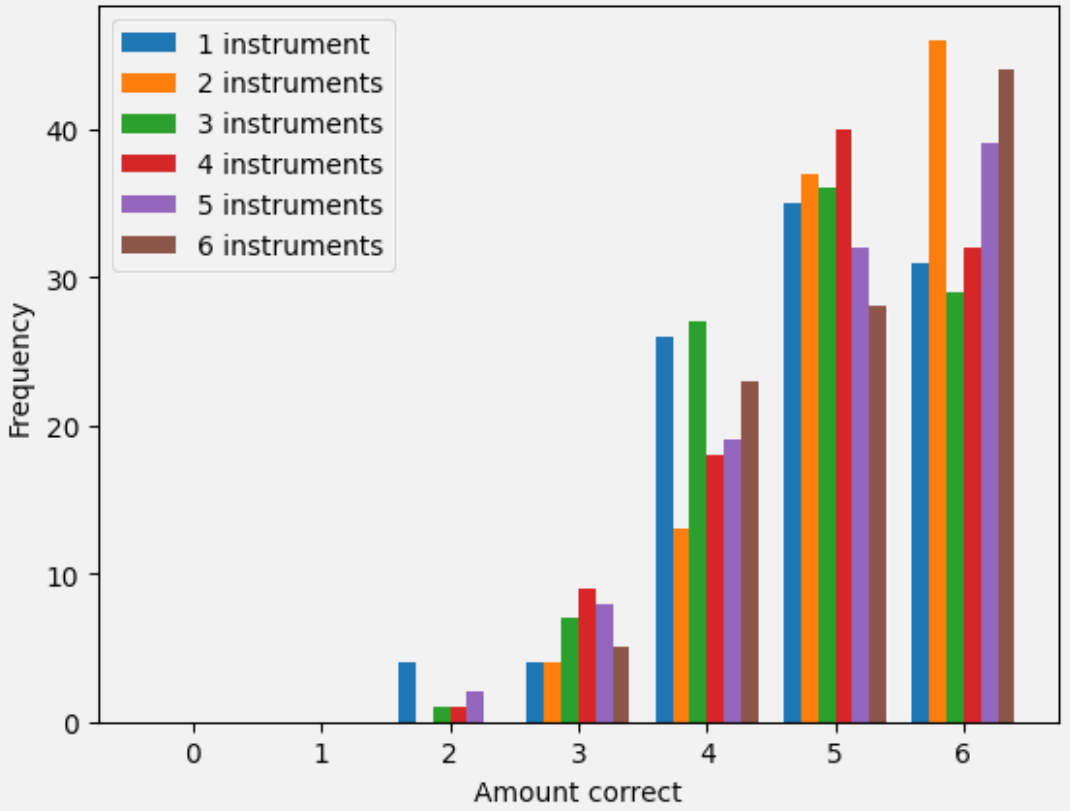
.wav input
Spectral input
Temporal branch (U-net)
Spectral branch (U-net)
Shared decoder layers
Skip conditions
Hybrid part between branches
Spectral output
Separated .wav output



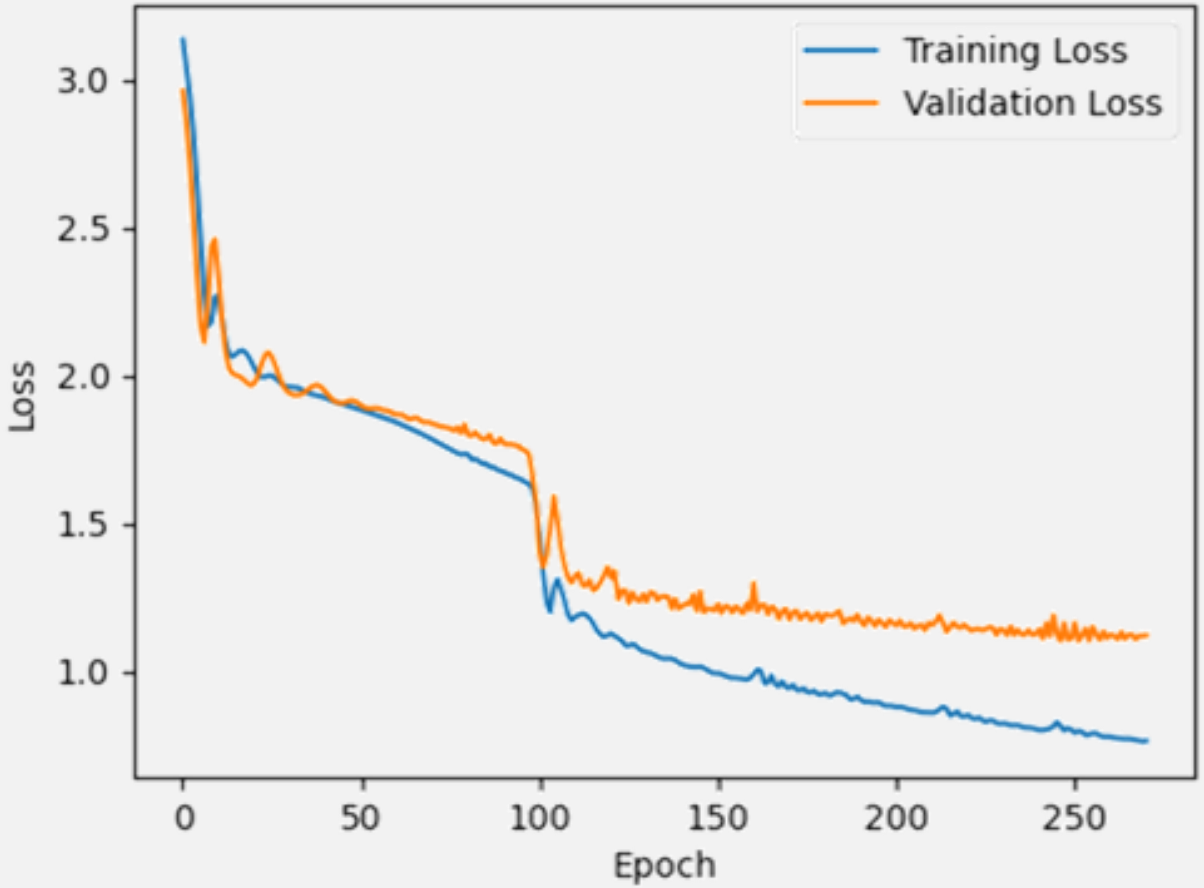
C. Extra slides with results

1. MLP instrument specific performance





2. MLP getting stuck



The MLP network tended to predict specific instruments to just not be present for all inputs in the training set for the first epochs. In this graph, you can clearly see that the network got ‘unstuck’ around epoch 100. With some hyperparameter optimization and adjustments of neurons per layer a stable and converging model was created.