University of Derby

Department of Electronics, Computing & Mathematics

A project completed as part of the requirements for

BSc (Hons) Mathematics and Computer Science

entitled

Automatic music transcription: Can Neural Networks be used to analyse audio for generating drum kit notation?

by

Daniel Topping
dtopping256@gmail.com

May 2019

Abstract

Currently, the way that a musician would transcribe music is manually by ear and then written up either on paper or with some notation software like Sibelius. This can take large amounts of time and can be prone to error; although currently, human experts are better at this than the state of the art Automatic Music Transcription (AMT) systems. The focus of this dissertation was on investigating the effectiveness of neural network models at recognising combinations of acoustic drum kit hits.

Recordings of 5 acoustic drum kits were used for raw input data. This was then labelled, processed and went through various augmentation steps to produce a large, synthetic data set of multi-classed hit combinations. This synthetic, multi-classed data was then used to train 3 neural network models to varying degrees of success, with the assumption that the prediction behaviour of the models with synthetic data would be similar to predictions on real data.

The model which was most accurate, with synthetic data, was Model B. In particular, the configuration with, one-hot encoded outputs and Kullback-Leibler divergence loss reached 83.8% class accuracy over 10 epochs. Then, a larger, final model was made based on the design of Model B. Kullback-Leibler divergence loss didn't scale very well with larger models, so binary cross-entropy loss was used instead; since this gave a similar performance during testing. The final model reached 92.4% overall accuracy across the 10 drum kit classes, which is nearly as good as an existing method that reached 99% across 3 drum kit classes.

Acknowledgements

I would like to thank Dr Dave Voorhis for supervising this dissertation, providing me with useful insight and feedback. Also, I would like to thank my mother for being patient enough to proof-read this and my family and friends for their support during my time studying.

Contents

1 Introduction

1.1 Project rationale

AMT is the process of analysing audio data and creating a human readable interpretation of what was played over time. Currently, human experts are better at transcribing music than the best AMT systems (Benetos et al. 2013); as there are many components to creating detailed transcripts and these need to be done to high accuracy. Aspects of this have been implemented successfully by Lunaverus (2019) with their AnthemScore program. AnthemScore uses a convolutional neural network to recognise which musical notes to write for certain frequency bands and how long the note is sustained. However, it isn't perfect and allows for manual corrections to the transcripts. Furthermore, it doesn't distinguish between different instruments, this results in a composite score of all of the detected instruments being made.

A drum kit is an unpitched instrument, comprised of drums and cymbals that can be played to produce polyphonic rhythms called drum beats. Each drum/cymbal will have a unique timbre and different combinations of these will produce a different overall texture of sound. This paper focused on the development of neural network models with the purpose of learning the timbre of the various parts of a drum kit. This was so that as well as being able to determine which drum-cymbal combinations were hit, it could distinguish between different techniques that were used; capturing more musicality from the audio information.

The benefits of neural networks that are able to recognise timbre, are that better AMT systems can be created, which have the potential to split songs into different scores for each instrument. This can be applied to other areas in sound and music as well, such as cancelling out the sound of instruments during music production, or generating artificial instrument timbres as shown by Google DeepMind's WaveNet (Oord et al. 2016). This demonstrates that there are plenty of commercially viable applications for similar models in this area of industry.

1.2 Aims & Objectives

The aim of this paper was to demonstrate that neural networks can be used to help create musical notation for drum kits.

Figure 1, shows most of the drums and cymbals that can feature on a drum kit. However, some are less common (such as the splash and china cymbals) so we don't have enough data to run reliable tests with them. For this reason, we will specifically look at: bass drum, the 3 tom drums, snare drum, crash cymbal, ride cymbal and hi-hat. As well as which drum/cymbal was hit, in transcription, it is often important to recognise the technique that was used. In the context of this report, we will be distinguishing between 'open' and 'closed' hi-hat as well as 'normal' and 'bell' hits of the ride cymbal when defining classes.

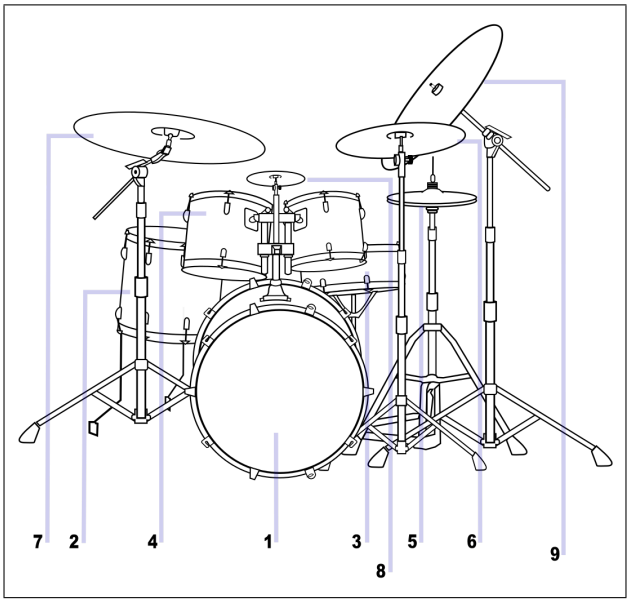


Figure 1: Drum kit components: 1 - Bass drum, 2 - Low (floor) tom, 3 - Snare drum, 4 - Mid tom, 5 - Hi-hat, 6 - Crash cymbal, 7 - Ride cymbal, 8 - High tom, 9 - China cymbal. Image from (Hashmi 2013).

The objectives are:

- Investigate different neural network models for learning drum kit timbre.
- Show that differentiating timbre is a trainable feature for neural network models.
- Train a neural network model to a high degree of accuracy.
- Apply this model to predict drum beat sequences, with pieces of untested audio.

2 Literature Review

2.0 Introduction

This literature review is split into 4 main sections: neural networks (section 2.1); sound analysis (section 2.2); previous work in this area (section 2.3) and refined research questions (section 2.4).

In the first section, there is a brief overview of neural networks and the back-propagation technique, as this is fundamental to the aim of this dissertation. There is a larger emphasis on convolutional neural networks because these are the current state of the art neural network models. Also, various model design patterns and structures which have been discovered during the work on GoogLeNet, ResNet and WaveNet are covered; since features of these were implemented in the design of the models used in the methodology. This section concludes with the obstacles to training machine learning models, with potential solutions to some of them.

In the second section, research into various sound analysis techniques was explained, to gain some understanding of how to measure timbre and how to create audio spectrograms. Also, this section covers onset detection which was used for making scripts to automate pre-processing.

In the previous work section, an alternative method of drum kit transcription using non-negative spectrogram factorisation was investigated. It was shown that this method was very reliable at detecting and classifying drum hits in polyphonic music, although it only studied 3 parts of the drum kit.

The literature review finishes with some refined research questions, which will be investigated as multivariate configurations of the models in the methodology.

2.1 Neural networks

Artificial Neural Network (NN) models are models loosely based upon the biological processes which happen in the central nervous system of complex organisms. They are models designed to approximate a function which represents a pattern in some data (Russell and Norvig 1995). For the purpose of this report, we will use NN models for supervised learning, although there are other ways NNs can be trained.

2.1.1 Biological neurons

Neurons are useful as inspiration for data processing models because each neuron performs a decision. Neurons get excited by the input from other neurons, which travel across many synapses to the neuron. If this combined excitation reaches a threshold point, then the neuron fires a signal which propagates down its axon to the synapses of other neurons. But, if it doesn't reach this threshold it doesn't fire (Rojas 1996, p.18-20).

2.1.2 Artificial neurons

The behaviour of neurons has been known to have useful applications in computing since McCullock and Pitts (1943) published their famous paper. They explained that the characteristics of neurons made them useful in 'propositional logic', which is useful in computing because neurons can be utilised to make decisions when given data. During this time,

simplistic neuron models which resembled basic logic gates were used to solve problems (McCullock and Pitts 1943, p.99). However these models were quite restrictive, only using binary information as input/output and were bespoke for the problems they addressed, so they weren't reusable for slightly different problems (Rojas 1996, p.29-33).

2.1.3 Perceptrons (early neuron models)

The 'Perceptron' was introduced by Rosenblatt (1958). This model improved upon previous neuron models by adding weights to each input connection (see Figure 2). This was done because he observed that using probability theory would be a better way to investigate the brain than boolean logic (Rosenblatt 1958, p387-388). This meant that learning algorithms could be used on a single perceptron, by adjusting the connection weights incrementally, to optimise towards a desirable output (see section 2.1.5). This could be done without having to change the overall structure as you would with McCullock-Pitts neurons. However, single perceptron models still had the downside of not being accurate when approximating problems where the decision boundary isn't linear (Raschka 2015). For example, the XOR problem cannot be solved by a perceptron because that problem requires the decision boundary to be of a higher order than linear (Rojas 1996, p.124-127).

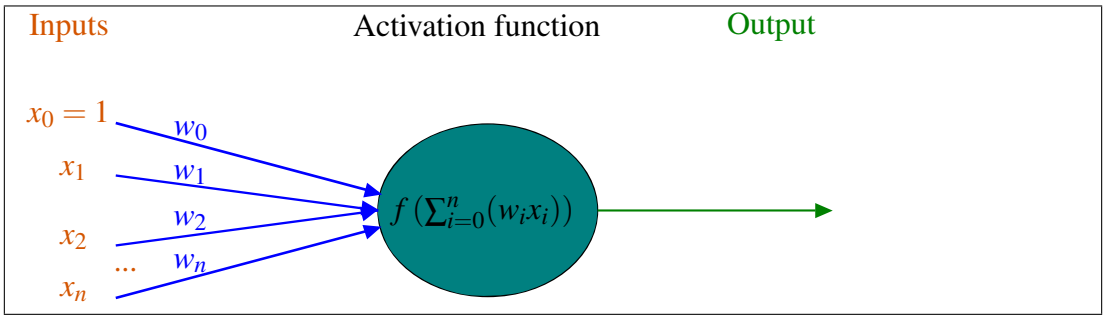


Figure 2: This is how a perceptron works. The sum of the products of the input values and their respective weights are put through an activation function, the result is the output. It is conventional to pass 1 into the first input x_0, along with the variable inputs so that the perceptron can still produce a non-zero output when the variable inputs are zero. The weight w_0 is called the bias.

2.1.4 Activation functions

Activation functions, which can be linear or non-linear, map the sum of the input signals (which have a linear domain) to some range (Nwankpa et al. 2018). Non-linear activation functions are crucial to neuron models because otherwise, they would only be capable of producing linear output, which restricts the usefulness of the model (Russell and Norvig 1995, p.567-568).

There are many types of non-linear functions which can be used as activation functions. More recently, there are 3 groups of activation functions that are widely used: Sigmoid, hyperbolic and rectified linear units (ReLU) (Nwankpa et al. 2018). ReLU functions were found to perform well for NN models built for audio, in particular, Leaky ReLU which is

a variation of ReLU where the function has a slight positive gradient for the negative part of the x-axis instead of being 0 as with regular ReLU (Maas, Hannun, and Ng 2013).

2.1.5 Back-propagation, loss functions and initial weights

The back-propagation algorithm adjusts weights of the inputs for each neuron incrementally so that the error (loss) is minimised. A loss function is a continuous function of the error between a prediction output of the model and the expected output that represents the actual class. This must be continuously differentiable so that optimisation algorithms such as gradient descent can minimise the loss function (Russell and Norvig 1995, p.580-581).

Two functions which can be used as loss functions are binary cross-entropy (BCE) and Kullback-Leibler divergence (KLD) which originate from information theory (MacKay 2013, p.34, p.67). Research has indicated that the initial weights of neural network models has an affect on the overall effectiveness of learning. For models using hyperbolic activations, Glorot and Bengio (2010) inspired 'Glorot normal' distributions for selecting initial weights. Similarly, for ReLU activations, He et al. (2015b) inspired the 'He normal' distributions for selecting initial weights. These loss functions and weight initialisers are provided by the Keras (2019) library.

2.1.6 Neural networks

NN models contain multiple neurons which are interconnected, to create more complexity. These are capable of understanding more sophisticated patterns than single neuron models. In dense (fully-connected) NNs, each neuron in a layer is connected to each neuron in the next layer. Dense NNs are good when modelling problems where the optimal structure is unknown because over time the network will eliminate unused connections by reducing the connection weights to 0. However, although these networks can be largely successful, they tend to be resource intensive; containing large amounts of unnecessary weights.

Sparse NNs are structured to improve performance by removing many connections which are assumed to have a negligible contribution to the model.

2.1.7 Convolutional neural networks

Convolutional Neural Network (CNN) models are neural networks which came about after Hubel and Wiesel (1959) did their work on the visual cortex of cats. This work inspired a structural design of neural networks where neurons only took inputs from small regions of data, making them more sparse and temporal. Also, it was observed that cats have areas in their vision that are sensitive to light as well as areas that are desensitised, allowing certain features to be picked up such as edges (Hubel and Wiesel 1959, p574). CNN models use regions that use the same group of weights called 'kernels' to store patterns, which filter features in a similar way.

Convolution layers map small regions of space from the proceeding layer to the next layer. These regions are manipulated by a convolution kernel which is a transformation of the region. Convolution layers can be used to increase the feature space of the data by passing the same regions through many different convolution kernels. Convolution layers are usually interleaved by pooling (sub-sampling) layers (Lecun et al. 1998). Pooling layers aim to simplify the data by down-scaling it in the sample space (see Figure 3),

although convolution layers with a stride of greater than 1 can be used to achieve the same thing (Springenberg et al. 2014). A famous CNN is LeNet-5 (Lecun et al. 1998) which was used for handwritten digit recognition.

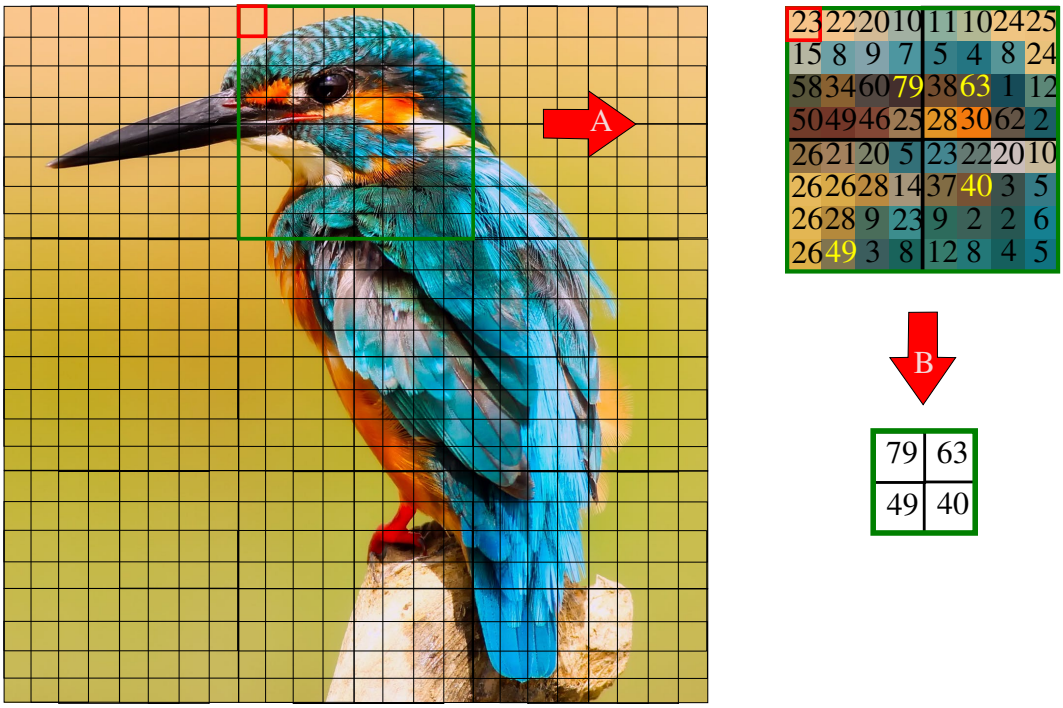


Figure 3: This figure demonstrates how data is convoluted and then pooled. A: The convolution process transforms the data using a convolution kernel in sections. B: The pooling process scales down data using some selection method (in this case max-pooling uses the greatest value). In practice, this is done with many different kernels and CNNs use these in layers repeatedly to efficiently extract features from the input. *Kingfisher picture from (12019 2017)*

2.1.8 Using more convolution layers instead of pooling layers

Springenberg et al. (2014) pointed out that by only using convolution layers in CNNs, less data is lost than with the more traditional CNNs which used pooling. The fully-convolutional NN has become a more popular approach since this observation, with the down-scaling of the sample space done by using convolution layers with stride 2, instead of the typically used max-pooling layers.

2.1.9 Causal and dilated causal convolution layers

In images, it makes sense to assume that the inputs around a given input have some relationship to that input since the dimensionality of the input is 2 space dimensions. However, with audio data, this assumption doesn't make sense, because time is a dimension. Instead, we assume that for a given input, it only has a relationship to inputs in the past from it. This is the idea behind the causal convolutions used by Google DeepMind's state

of the art WaveNet (Oord et al. 2016), where neurons are only given inputs from up until present in the sample space. Dilated causal convolutions increase the receptive field of the neurons while using the same amount of inputs by spacing the connections between each layer (Oord et al. 2016). Both of these layers were shown to be useful in creating sparser CNN's, which increased computational performance as well as the accuracy of the models.

2.1.10 Non-sequential NN designs

Modern NN's can be very complex and don't follow a completely sequential design pattern. For example, GoogLeNet uses frequent branching and concatenation of different sized convolutions in its 'Inception modules'. These are used in a repeating pattern in their deep CNN (Szegedy et al. 2015). It was also found that convolution layers with a 1x1 kernel were a good way to reduce the size of the feature space, without affecting the size of the sample space. This is used in GoogLeNet's Inception modules, before each convolution layer with a larger kernel.

Microsoft's ResNet demonstrated that when NN's get to a certain depth they can become less accurate, so proposed skip connections; which allow earlier activations to pass deeper into the NN (He et al. 2015a). This was important because before ResNet it was widely believed that the deeper the NN was, the better it performed.

2.1.11 Machine learning obstacles

i Multi-classed data

Many machine learning models are designed to use data with a single class per piece of data, with classes which are mutually exclusive. This data can easily be labelled[1] one label per class using a one-hot encoded output. For example, the MNIST data set has pictures of handwritten numbers with labels valued 0-9 (LeCunn 2010). Only one of these labels is true per image since the classifier is the number that the handwritten digit represents which is a mutually exclusive classifier. However, classifying drum kit timbre means analysing sound that could contain multiple drums or cymbals simultaneously. The presence of these classes are not mutually exclusive so the data is multi-classed.

There are multiple ways of encoding multi-classed data. The multi-hot method is where each class is assigned to one label, each is encoded as 'on' or 'off' depending upon the presence of the class. The one-hot method can also be used, in this context each label is assigned to a super-set, representing some combination of classes with only one label encoded as 'on' and the rest are 'off' (Maxwell et al. 2017). For drum kit classes we will see that for the 10 drum/cymbal classes, there are 87 feasible hit combinations. This means that with one-hot encoding there are 87 labels and with multi-class encoding, there are 10 labels.

An activation function in the output layer needs to be used which will normalise the output so that it is within the domain of the class encodings. From the report by Maxwell et al. (2017), it seems that the Softmax function is better suited to output with one-hot encoding and the Sigmoid function is better suited to output with multi-hot encoding. Their reasoning for this is that the sum of Softmax output is 1, so it is easier to train one-hot encoded classes with Softmax and harder to train multi-hot encoded classes. Sigmoid

[1] A marker associated with some data, which classifies it.

functions were found to perform better than the ReLU functions and hyperbolic functions for the final activation layer of a multi-hot encoded model, which was designed to predict combinations of 3 diseases using medical records (Maxwell et al. 2017).

ii Underfitting, overfitting and generalisation

As well as underfitting, where a machine learning model hasn't learnt enough from training on a data set, overfitting is also a problem. Overfitting is when the machine learning technique used to approximate some unknown model, creates an over-complicated model; which is very accurate to the training data, but not accurate to data which it hasn't encountered before (Rojas 1996, p.143-145). Optimally, models will generalise so that they can make accurate predictions on new and diverse data sets. A common regularisation technique is dropout, where random neuron connection weights are set to zero during training (Hinton et al. 2014).

iii Obtaining training data

While it is possible to create a labelled data set manually, this can take large amounts of time and physical storage; especially if you want a high accuracy NN because more data is typically better. A solution to this is data augmentation (Wong et al. 2016), where incremental transformations can be used to create new data and reduce overfitting (see section 2.1.11 part ii). Wong et al. (2016) found that it is best to use data augmentation to create transforms in the data space (before getting processed by a NN), rather than mid-way through in the feature space. Data augmentation can be done as a part of pre-processing or on the fly during training, depending upon the restrictions of the machine.

Another aspect of data collection that is particularly relevant for classification problems, is that certain classes tend to naturally occur more frequently than others. This is worse for smaller data sets because it becomes less feasible to remove the difference in data from the larger classes, as this reduces the size of the data set even more. The reason why this is a problem for NN models is that it will be less accurate at predicting classes with a smaller proportion of the input data. Synthetic Minority Over-sampling Technique (SMOTE) is a method proposed by Chawla et al. (2002) to address the problem of imbalanced data sets. It involves creating new unique samples of the minority class in the same way that data augmentation creates new unique samples. It was shown that this approach improved the classification accuracy of minority classes (Chawla et al. 2002).

Another problem is, what are the correct proportions of training, validation and test data to use when testing a model? Too much validation and test data can impede the overall training of a NN, and on the other hand, too little means that the metrics which measure performance are less accurate.

2.2 Sound analysis

Timbre is the information in a sound which indicates to the listener what is producing the sound. Usually, timbre is used to describe what instrument is producing the sound; we will use timbre to distinguish what drum or cymbal is hit and when.

Timbre differs to overall pitch because one instrument can produce a C note while another can also produce a C note, however, they both sound different due to the materials and how the sound is created.

Timbre also encompasses the duration's of various structures in the waveform[2] of the sound; such as attack, decay, sustain and release, which are variables edited in sound production using music software (Müller 2015, p.26-29). For the context of the drum kit, only the attack and decay phases are considered since the sustain and release will require many quick consecutive hits to produce and we're only interested in single hits over a very small amount of time.

2.2.1 Fourier transforms

A Fourier transform of a sound separates a complicated wave into its component frequencies. This technique is usually used to determine the note of pitched instruments such as a piano, however even though drums and cymbals are said to be unpitched, they still resonate at different frequencies to one another and this could be important in differentiating the timbres. In particular, the pattern of overtones for each component of the drum kit could be useful in distinguishing them.

From work done on analysing the difference in the sound of violins by Yokoyama, Awahara, and Yagawa (2016), it was shown using fast Fourier transforms, that playing style can also affect the overtones of the instrument. These can be used to detect playing technique as well as the instrument itself, which will be useful for the transcription of the technique for the drum kit.

Fourier transforms can be done on a whole section of audio to generate a measure of frequency against magnitude for the whole segment. However sound changes over time, so to take the shape of a sound into account short-time Fourier transforms can be used to yield 2D spectrograms of the magnitude of the frequency over time (Müller 2015, p.53).

2.2.2 Onset detection

Speed of music (tempo) is an important feature of transcripts, telling musicians how quickly music is playing. In the context of sound processing, onset detection enables us to evaluate the speed of a beat and gives a measure of how to break up music into parts. This will help place a drum or cymbal hit into a position in music (Müller 2015, p.304).

Energy based novelty is one way of finding onsets[3], by using window functions to map a signal to localised energy over time and then find the log difference of this energy over time (Müller 2015, p306-307). In the resulting output, noise can be ignored and the actual hits can then be selected by applying an adaptive threshold function (Bello et al. 2005).

2.3 Work already done in the area of drum transcription

J Paulus and Virtanen (2005) used features taken from spectrograms of drum sounds and compared these with positive template spectrograms created from an average of some test data. The model had a 99% 'precision rate' for unprocessed data, although it was only used with 3 parts of a drum kit (hi-hat, snare drum and bass drum) which all sound very different and didn't attempt to classify different technique.

[2]Defined as "a usually graphic representation of the shape of a wave that indicates its characteristics (such as frequency and amplitude)", by Merriam-Webster (2019).

[3]The point in time at the start of a transient (sudden excitation of a waveform) (Bello et al. 2005; Müller 2015)

2.4 Refined research questions

Although the research has provided a much clearer picture of how to meet the objective, it also raises certain unanswered questions.

Input data format
Which format of input data is better for neural network accuracy, input data presented as 1D amplitude data over time or as 2D frequency density spectrogram data?

Output data encoding
Is expressing the output as a multi-label (multi-hot encoded) problem or a super-class (one-hot encoded) problem, better for neural network accuracy?

Loss function
Which loss function is better at training the neural network?

3 Methodology

3.0 Introduction

This methodology explains the details of the implementation of the training, improvement and testing of 3 CNN models which were designed using elements of what was learned from previous work covered in the literature review (see section 2).

The Keras (2019) Python library was used as a front-end to define the models, with TensorFlow doing the work on the back-end. This was done because Keras is more high-level than TensorFlow, so defining a NN is more legible and concise than using TensorFlow by itself and TensorFlow has been used by many researchers and companies as it is a powerful framework on its own. TensorBoard (which comes with the TensorFlow package), is also a useful tool as this makes logging data from TensorFlow callbacks easy and can produce detailed diagrams for the structure of the models.

When designing the models, some of the techniques found in research didn't always work when applied to this use case or had to be adapted because of the constraints of the computational power available; so the design methodology was incremental.

To resolve the refined research questions (see section 2.4); models A and B used 1D amplitude input data and Model C used 2D spectrogram input data with the encoding of the output data and loss function being varied during training. When using one loss function to train the network, the others were used as metrics to aid in comparing the models, as well as a label specific accuracy (categorical accuracy or binary accuracy depending upon the encoding) and a class accuracy metric which can be used for all of the models.

There is a large focus on how the input data was obtained in this methodology because NN models require large amounts of diverse data to work well.

To summarise the whole process:

1. Raw wave file data was obtained and pre-processed, so that it was a consistent format for the CNN models to train upon and there was plenty of natural variation.

2. The amount of data was increased with data augmentation techniques and multi-class data was efficiently generated and labelled; using the assumption that training using synthetic multi-class data will produce a similar accuracy when predicting real multi-class data.

3. The synthetic data was then divided into training, validation and testing data sets.

4. The models were designed and adjusted by training them on a smaller subset of the training and validation data sets.

5. Then for each working model, it was trained using the full data set and values of loss, accuracy and other relevant metrics per epoch[4] were saved to a log file.

6. The best model was improved and used to predict the drum/cymbal hits of data from a real beat and the accuracy was compared.

[4] A full run over all of the training and validation data

3.1 Obtaining input data

3.1.1 Recording, processing and labelling

For neural networks to be effective at learning generalised patterns, we need the data set to be as diverse as possible (see section 2.1.11, part ii). There were 5 different drum kits which were recorded with 3 different pieces of recording equipment and 3 different players. Each of these factors had an effect on the overall sound of the drums played, adding to the diversity of the data.

For each drum kit, the following was recorded:

- *Isolated hits*: Many slow repeated drum/cymbal hits in isolation (which are single-class), which were used for the development and benchmarking of the models.
- *Drum beats*: Drum beats with combinations of drum/cymbal hits (which are multi-class), which were used for evaluating the accuracy of the final model.

Both categories of audio were manually converted into a single channel WAV, with just the isolated hits additionally being labelled by putting them into a sub-directory structure of the following pattern: hit type, kit type and technique type; which describe: the object used to hit the drums/cymbals, the drum or cymbal being hit and the technique used respectively (this is covered in detail in section 3.1.3).

To get equal shapes of input data (as each recording device had a different sampling rate) and to reduce the size of the information for the efficiency of the augmentation process as well as the neural network, the isolated hits were re-sampled to 48kHz using a function from the SciPy package (SciPy 2019). It was down-sampled further, for efficiency to 3kHz and a custom onset detection function (see section 2.2.2) using an energy-based novelty (EBN) window function and an adaptive threshold function (see Figure 4) were used to find where the drum waveforms start and the audio was cropped from the 48kHz version, into short snippets of 0.25 seconds (12000 samples). Each crop was saved separately in a compressed GZIP format, to reduce the physical storage taken up for each stage of processing.

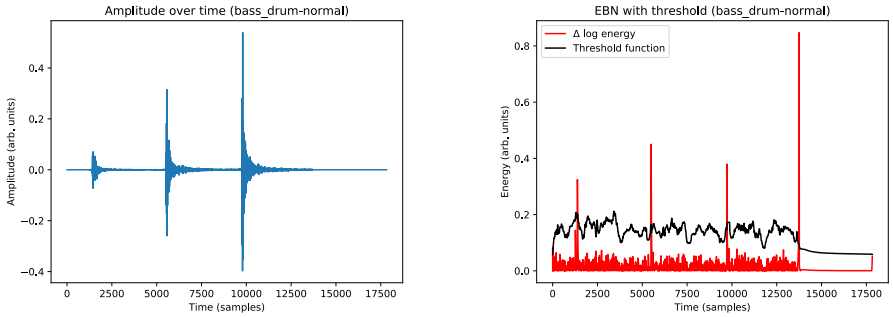


Figure 4: On the left, is the amplitude over time of a recording of 3 bass drum hits. On the right, is the EBN against an adaptive threshold function, where onsets are taken to be the first point above the threshold with an upward gradient. This method is used to detect the start of transients in audio, but will also detect unwanted transients where audio is suddenly muted (during manual processing) and will make peaks at either end due to the window function. The peak-picker identified the peaks at 1382, 5494 and 9722 which are correct, as well as, 13753 which is where the background noise is suddenly muted.

3.1.2 Data augmentation

By this stage, 1069 audio segments were collected, but neural networks require a lot more data to be effective. So data augmentation (see section 2.1.11 part iii) was used to create more data by transforming existing data.

Data augmentation transformations were used before and after the single-class data was made into multi-class data (see Figure 5). These were each applied in 10 different amounts per augmentation per crop and in all possible combinations, while making sure that the sound isn't unrecognisable, ensuring that the label is still valid.

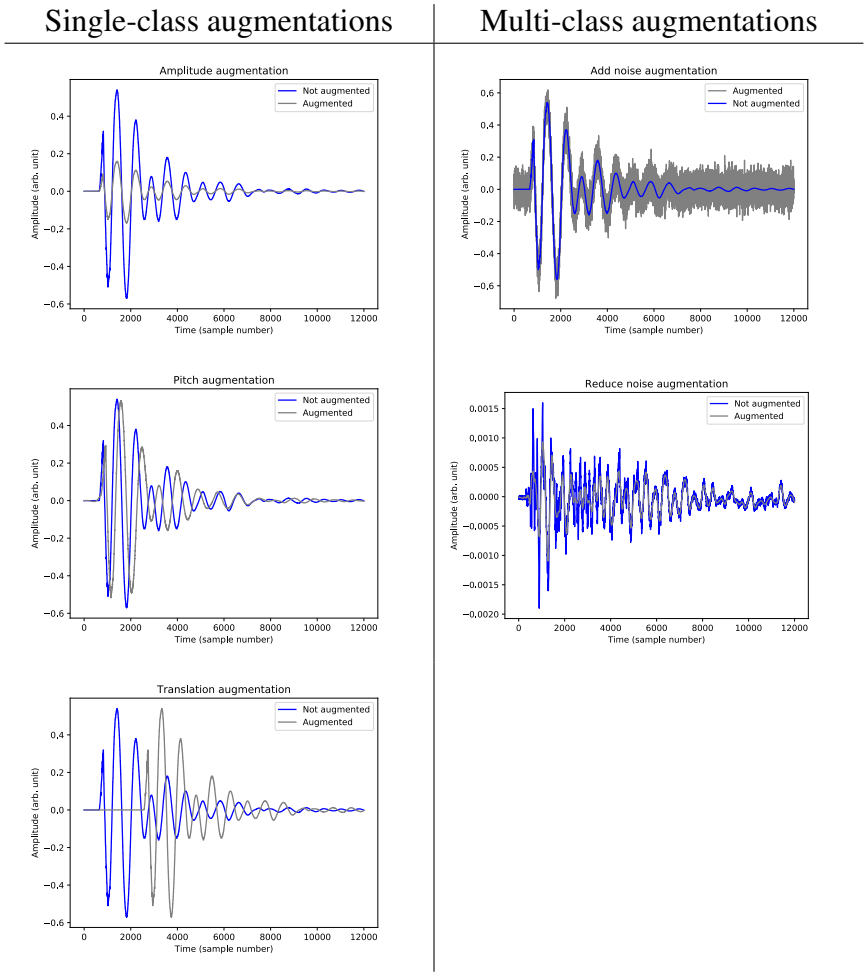


Figure 5: Plots of the individual data augmentation transformations on bass drum waveforms. For each plot, the original waveform (in blue) is against the transformed waveform (in grey); with sample number on the x-axis and amplitude on the y-axis.

The single-class augmentations aim to diversify the data for how a drum or cymbal could sound. Pitch is one way of altering the sound while preserving its class. Pitch varies naturally depending on the materials used to make the instrument, the shape of the instrument, the tension of the drum skins, etc. Similarly, drums and cymbals can be played softer or louder relative to one another so the amplitude is augmented. Lastly, drums and cymbals can be played in slightly different orders since they won't occur at precisely the same time, so the translation along the time axis of the waveform is augmented. The

NumPy package, included with SciPy (2019) and the Librosa (2019) package was used for transforming the data.

For each valid combination of classes, augmented single-class waveforms were superimposed by taking the sum of the amplitude values together element-wise, producing multi-class data. Augmented single-class data was reused multiple times for making multi-class data, but was not used in the same combination; which guarantees that each one is unique.

Here an assumption is made, that superimposing the single-class data in this way creates synthetic multi-class data that is very similar to real multi-class data. However, since two recordings can come from different places with varied sound quality and acoustics this is just an assumption. Even so, it can be observed that waveforms made this way do strongly resemble real waveforms (see Figure 6).

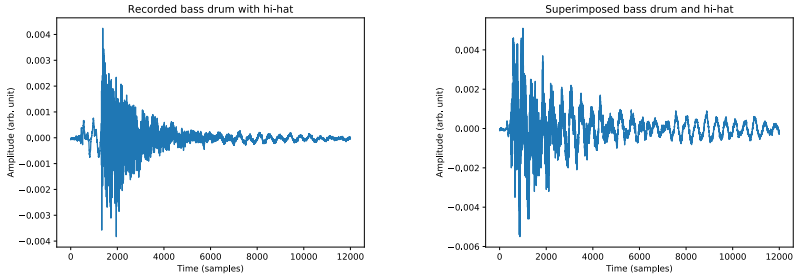


Figure 6: This compares an actual recording of a bass drum and hi-hat hit together (on the left), with a superimposed bass drum and hi-hat hits which were recorded separately (on the right). As you can see the waveforms look very similar. We can see a difference in translation since the superimposed hits start together and in the actual recording, there is a lag of a fraction of a second between where the drummer plays the bass drum followed by the hi-hat. Also, there is a change in relative amplitude of bass drum and hi-hat between the graphs, as the waveform of the hi-hat is a lot more clear and less distorted by the bass drum with the recorded version than with the superimposed version.

After the data is made into multi-class data, it makes sense for augmentations involving adding white noise and reducing noise to be applied uniformly. This is because noise won't vary with the instrument, but with the recording device.

At the end of processing, there were in total 132,243 files that could be used for training taken from various stages of this process (see Figure 7), which is a vast improvement from 1069 at the start.

Processing stage	Audio files
Single-class with no augmentation	1069
Single-class with APT augmentations	31868
Multi-class with no augmentation	188
Multi-class just APT augmentations	8689
Multi-class just noise augmentations	2140
Multi-class with all augmentations	116760

Figure 7: Number of files collected for the input data from each stage of processing. APT augmentations = amplitude-pitch-translation augmentations. Note: variants of the data were made which skipped certain steps, such as: 'multi-class with no augmentation' and 'multi-class just noise augmentations'.

3.1.3 Labelling and encoding

To label large quantities of data efficiently, sub-directories were structured in the following way: hit types, kit types and technique types (see Appendix A.1). This was done so that multiple characteristics of a hit could be labelled and this would represent one class. When the data became multi-classed after isolated hits were superimposed (see section 3.1.2) these labels would merge.

Classes were encoded during training, by a custom generator object which inherits from the Keras generator. This reads the labels from the audio file-path and encodes them as one-hot or multi-hot:

One-hot encoding
For this case, a matrix of size 87 (number of combinations) is produced, where the j^{th} bit is 1 and the rest are 0, with j being the index of the hit combination label (see Appendix A.2).

Multi-hot encoding
For this case a matrix of size 10 (number of classes) is produced, where a label is 1 if the drum/cymbal class is present in the combination. Note: for valid combinations, a maximum of 3 1's will be present at any given multi-hot matrix.

3.1.4 Spectrograms

The generator object was also responsible for converting the audio into spectrograms (when necessary) in real-time. Spectrograms would be used for some of the models to compare their performance to other similar models using 1D amplitude. For creating the spectrograms, the spectrogram method of the SciPy package was used; which uses short-time Fourier transforms (see section 2.2.1) to create 2D linear spectrogram output. It was found that to closer represent how humans hear, this could be made logarithmic; this had the effect of increasing the detail of the overtones shown but also increased the amount of noise in the spectrograms (see Figure 8).

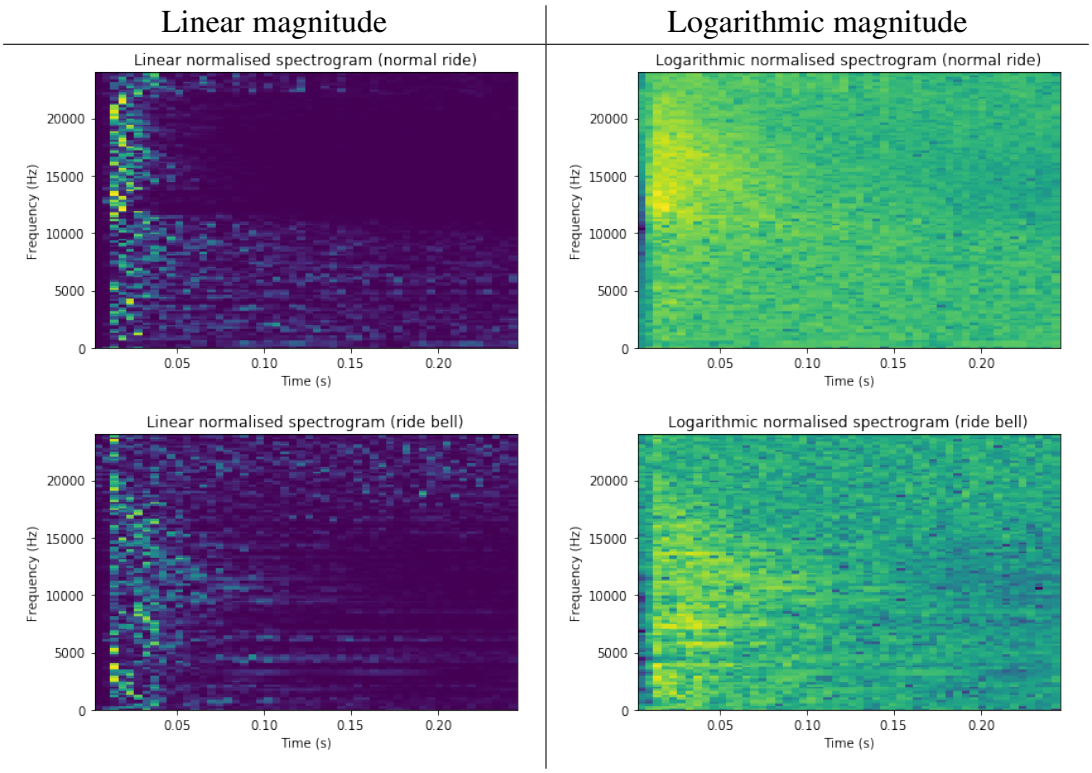


Figure 8: Here are the 129x53 spectra of two ride cymbal hits. All have been normalised and represent the data that is given to the 2D CNN models. Linear magnitude is much less sensitive to small changes than logarithmic magnitude. This means logarithmic magnitude suffers from having a lot of noise, however on the plus side, logarithmic magnitude exaggerates differences between timbres.

3.1.5 Dividing input data into training, validation and test data sets

The rationale behind dividing input data in this way is to have a measure of whether the model is underfitting, overfitting or is generalising (see section 2.1.11 part ii). The more data allocated to training the network compared to validation and testing, the less reliable the accuracy measure is for validation and testing; however, the network needs as much training data as possible to make accurate predictions in the first place. For this reason, the data was split 3:1:1 into training, validation and testing respectively.

3.2 Model design

There were 3 deep CNN models designed. For simplicity, they are referred to as Model A, Model B and Model C.

All of the models used the following design principles, which were covered in the literature: start with convolution layers of a stride greater-than 1, to reduce the size of the sample space (see section 2.1.8); every so often use dropout layers in the structure to randomly sever network connections during training (see section 2.1.11 part ii), which increase in drop-rate[5] as dropout layers occur deeper into the network; use 1x1 convolution layers instead of pooling layers to reduce the feature space (see section 2.1.10),

[5]The probability that a connection is severed by a dropout layer during training.

preserving detail; and at regular intervals use skip connections (see Section 2.1.10), so that older features have a greater effect on output.

As an overview, Models A and B analyse 1-dimensional amplitude float data in the domain of -1.0 and 1.0, while Model C analyses spectrograms of the same data, made using short-time Fourier transforms (see Section 2.2.1) with the SciPy python package. This data, in the form of spectrograms, have a different shape that is 2 dimensional instead of 1 dimensional. Model A has the largest number of layers compared to Models B and C which are similar in design except that C uses 2D convolution layers instead of 1D convolution layers. All models use skip connections, but only Model A uses inception modules (see section 2.1.10).

Models were made so that they could be run by a machine using the 'GeForce GTX 650' graphics card, this involved making sure that layer shapes were small enough to fit into less than 2GB of memory. For this reason, multiple 2 stride convolution layers were used at the start of each model, to down-scale the sample space, some models used a maximum of 32 filters and the batch size was 50. The processing speed of model training, was proportional to the number of layers in the model, that the GPU and CPU had to process. This is because each layer needs to wait for the previous layer.

3.2.1 Model A

This model takes 1D amplitude data in a $(B, 12000, 1)$ shape to make predictions. The batch size (B) used during training was 50. The shape of the output is (B, O) where O is 87 or 10 when the encoding is one-hot or multi-hot respectively. It makes use of causal convolution (CC), dilated CC, dropout, inception modules and skip connections. After CC layers, the Tanh hyperbolic activation function is used and for the final dense layer Softmax or Sigmoid is used as the activation function for one-hot encoding and multi-hot encoding respectively.

	Layers					Activation	Matrix shape	Amount
Initial down-sampling layers	CC 7/8 (s = 3)					-	$(B, 12000, 1)$	x1
	CC 7/16 (s = 2)					Tanh	$(B, 4000, 8)$	
	CC 5/32 (s = 2)					Tanh	$(B, 2000, 16)$	
	CC 5/32 (s = 2)					Tanh	$(B, 1000, 32)$	
	Dropout layer (rate = 0.1)					-	$(B, 500, 32)$	
Inception module	Skip	Dilated CC 3/32 (d = 2)				Tanh	$(B, 500, 32)$	x3
		CC 1/32	CC 1/1	CC 1/1	CC 1/1	-	$(B, 500, 32)$	
			CC 3/32	CC 5/32	CC 7/32	Tanh	$(B, 500, 1)$	
		Concat				Tanh	$(B, 500, 128)$	
		CC 1/1				Tanh	$(B, 500, 32)$	
	Matrix addition					-	$(B, 500, 32)$	
	Dropout layer (rates: 0.15, 0.20, 0.25)					-	$(B, 500, 32)$	
Flatten and dense layers	Flattened layer					-	$(B, 500, 32)$	x1
	Dense layer					Softmax or Sigmoid	$(B, 16000)$	
	Output					-	(B, O)	

Figure 9: A tabular representation of the structure of Model A, where: CC layers denoted with 'x/y' means x kernel size by y filters, 's' is the stride, 'rate' is the probability of a dropped connection and 'd' is the dilation rate. The matrix shape (a, b, c) shows the batch size (a), sample space size (b) and the feature space size (c); after the flattened layer, the sample space and feature space become the same dimension ($b \times c$). Note: for CCs where the stride and dilation rate are not specified, the stride is 1 and the dilation rate is 1 (no dilation). See Appendix B.1 for the TensorBoard diagram.

As shown by Figure 9, Model A makes use of 4 convolution layers with a stride greater-than 1 at the beginning, which reduce the sample space from 12000 nodes to 500 while adding 32 filters to the feature space.

Model A uses the hyperbolic Tanh function as its activation function between CC layers because Tanh was found to perform well with the WaveNet NN model (Oord et al. 2016), with the weights initialised from random numbers based on the Glorot normal distribution (see section 2.1.5).

Model A, has 3 lots of inception modules which split the activation of a 32 filter, dilated causal convolution layer into 4 distinct towers. For 3 of these towers, first, a convolution layer with a kernel size of 1 reduces the feature space to 1. This is then followed by another convolution layer with a larger kernel size (3, 5 or 7). For the other tower, a convolution layer of kernel size 1, with 32 filters is used. After this all of the towers' activations are concatenated along the feature axis which produces a layer with 128 filters and the sample space is kept the same. This is reduced back down to 32 filters by another convolution layer with a kernel size of 1 before being added to the activation from the skip connection. At the end of each module is a dropout layer which has a probability to sever connections during training, this increases from 15% in the first module to 20% in the second and is 25% in the final module.

After the inception modules, the CNN is flattened and put through a dense layer which has an output that is the same same as the encoded labels. The activation function throughout this model is Tanh apart from the final layer which is Softmax for one-hot encodings and Sigmoid for multi-hot encodings.

3.2.2 Model B

Like Model A, this model takes 1D amplitude data in a $(B, 12000, 1)$ shape to make predictions. The batch size (B) used during training was 50. Again, the shape of the output is (B, O) where O is 87 or 10 when the encoding is one-hot or multi-hot respectively. It makes use of CC, dilated CC, dropout and skip connections. For this model, a leaky ReLU activation function is used between CC layers and for the final dense layer Softmax or Sigmoid is used as the activation function for one-hot encoding and multi-hot encoding respectively.

	Layers		Activation	Matrix shape	Amount
Initial down-sampling layers	CC 7/8 (s = 3)		-	$(B, 12000, 1)$	x1
	CC 7/16 (s = 2)		Leaky ReLU (g = 0.3)	$(B, 4000, 8)$	
	CC 5/32 (s = 2)		Leaky ReLU (g = 0.3)	$(B, 2000, 16)$	
	CC 3/32 (s = 2)		Leaky ReLU (g = 0.3)	$(B, 1000, 32)$	
	Dropout layer (rate = 0.1)		-	$(B, 500, 32)$	
Stacked dilated CC layers	Skip	CC 1/1	-	$(B, 500, 32)$	x3
		Dilated CC 3/8 (d = 2)	-	$(B, 500, 1)$	
		Dilated CC 3/32 (d = 2)	Leaky ReLU (g = 0.3)	$(B, 500, 8)$	
	Matrix addition		-	$(B, 500, 32)$	
	Dropout layer (rates: 0.15, 0.20, 0.25)		-	$(B, 500, 32)$	
Flatten and dense layers	Flattened layer		-	$(B, 500, 32)$	x1
	Dense layer		Softmax or Sigmoid	$(B, 16000)$	
	Output		-	(B, O)	

Figure 10: A tabular representation of the structure of Model B, where: CC layers denoted with 'x/y' means x kernel size by y filters, 's' is the stride, 'rate' is the probability of a dropped connection, 'd' is the dilation rate and 'g' is the gradient of the negative part of the Leaky ReLU activation function. The matrix shape (a, b, c) shows the batch size (a), sample space size (b) and the feature space size (c); after the flattened layer, the sample space and feature space become the same dimension ($b \times c$). Note: for CCs where the stride and dilation rate are not specified, the stride is 1 and the dilation rate is 1 (no dilation). See Appendix B.2 for the TensorBoard diagram.

As shown by Figure 10, Model B makes use of 4 convolution layers with a stride greater-than 1 at the beginning, which reduce the sample space from 12000 nodes to 500 while adding 32 filters to the feature space.

Model B uses Leaky ReLU (see section 2.1.4) as an activation function between layers. The gradient of the negative part of Leaky ReLU was set to 0.3. The weights were initialised from random numbers based on the He normal distribution (see section 2.1.5).

Model B has 3 lots of sections which contain stacked dilated CC layers. This section starts with a causal convolution layer with a kernel size of 1 and 1 filter to reduce the feature space. Then this is followed by 2 consecutive dilated CC layers which are dilated at a rate of 2. After this, the activation of the last dilated layer is added to the activation of the skip connection. At the end of the section is a dropout layer which has a probability to sever connections during training, this increases from 15% in the first section to 20% in the second and is 25% in the final section.

After the stacked dilated CC sections, the CNN is flattened and put through a dense layer which has an output that is the same same as the encoded labels. The activation function throughout this model is Leaky ReLU apart from the final layer which is Softmax for one-hot encodings or Sigmoid for multi-hot encodings.

3.2.3 Model C

Model C is very similar to Model B but for 2D data. This model takes 2D spectrogram data in a $(B, 129, 53, 1)$ shape to make predictions. The batch size (B) used during training was 50. Again, the shape of the output is (B, O) where O is 87 or 10 when the encoding is one-hot or multi-hot respectively. It makes use of 2D convolution, dilated 2D convolution, dropout and skip connections. For this model, a leaky ReLU activation function is used between convolution layers and for the final dense layer Softmax or Sigmoid is used as the activation function for one-hot encoding and multi-hot encoding respectively.

	Layers		Activation	Matrix shape	Amount
Initial down-sampling layers	2D conv 5/16 (s = 2)		-	$(B, 129, 53, 1)$	x1
	2D conv 3/32 (s = 2)		Leaky ReLU (g = 0.3)	$(B, 65, 27, 16)$	
	Dropout layer (rate = 0.1)		-	$(B, 33, 14, 32)$	
Stacked dilated 2D conv layers	Skip	2D conv 1/1	-	$(B, 33, 14, 32)$	x3
		Dilated 2D conv 3/8 (d = 2)	-	$(B, 33, 14, 1)$	
		Dilated 2D conv 3/32 (d = 2)	Leaky ReLU (g = 0.3)	$(B, 33, 14, 8)$	
	Matrix addition		-	$(B, 33, 14, 32)$	
	Dropout layer (rates: 0.15, 0.20, 0.25)		-	$(B, 33, 14, 32)$	
Flatten and dense layers	Flattened layer		-	$(B, 33, 14, 32)$	x1
	Dense layer		Softmax or Sigmoid	$(B, 14784)$	
	Output		-	(B, O)	

Figure 11: A tabular representation of the structure of Model C. Where: 2D convolution layers denoted with 'x/y' means x kernel size by y filters, 's' is the stride, 'rate' is the probability of a dropped connection, 'd' is the dilation rate and 'g' is the gradient of the negative part of the Leaky ReLU activation function. The matrix shape (a, b, c, d) shows the batch size (a), 2D sample space size (b, c) and the feature space size (d); after the flattened layer, the sample space and feature space become the same dimension ($b \times c \times d$). Note: for 2D convolutions where the stride and dilation rate are not specified, the stride is 1 and the dilation rate is 1 (no dilation). The section with the stacked dilated convolution layers is repeated 3 times to increase the depth of the NN model. See Appendix B.3 for the TensorBoard diagram.

As shown by Figure 11, Model C is very similar in structure to Model B. One difference is that it uses 2 convolution layers with a stride greater-than 1 at the beginning instead of 4, which reduce the sample space from 129x53 to 33x14 while adding 32 filters to the feature space. This is because this 2D representation (input size: $129 \times 53 = 6837$) of the data already isn't as high in resolution as the 1D equivalent (input size: 12000), so requires less down-sampling.

Like Model B, Model C uses Leaky ReLU (see section 2.1.4) as an activation function between layers. The gradient of the negative part of Leaky ReLU was set to 0.3. The weights were initialised from random numbers based on the He normal distribution (see section 2.1.5).

Similarly, Model C has 3 lots of sections which contain stacked dilated 2D convolution layers. This section starts with a 2D convolution layer with a kernel size of 1 and 1 filter to reduce the feature space. Then this is followed by 2 consecutive dilated CC layers which are dilated at a rate of 2. After this, the activation of the last dilated layer is added to the activation of the skip connection. At the end of the section is a dropout layer which has a probability to sever connections during training, this increases from 15% in the first section to 20% in the second and is 25% in the final section.

After the stacked dilated CC sections, the CNN is flattened and put through a dense layer which has an output that is the same same as the encoded labels. The activation function throughout this model is Leaky ReLU apart from the final layer which is Softmax for one-hot encodings or Sigmoid for multi-hot encodings.

3.3 Model training

All models were trained using back-propagation by minimising their respective loss functions.

The loss functions and output label encoding was varied to capture every combination of these factors to ensure that they were tested fairly (see Figure 12). This was done to see which combination performed the best (see section 2.4).

	Encoding type	
Losses	One-hot & BCE loss	Multi-hot & BCE loss
	One-hot & KLD loss	Multi-hot & KLD loss

Figure 12: The 4 encoding-loss configurations which were run over each model.

On certain training runs only the optimisation algorithm was changed to see whether there was a significant difference in minimisation of the loss function, the algorithms tested were: Stochastic Gradient Descent, Adagrad and Adam. Also for Model C linear spectrogram input was compared to logarithmic spectrogram input to see which gave the best accuracy. GPU processing was enabled which significantly increased processing speed, however, this meant reducing the size of the models and using smaller batches of input data so that the GPU didn't run out of memory.

3.4 Model analysis

For analysis of how well models performed the Keras (2019) API was used to calculate accuracy metrics for the models. Also to quantitatively measure common mistakes of the models, confusion matrices were produced.

3.4.1 Accuracy metrics

During model analysis, an 'all or nothing' (AON) class accuracy was a consistent metric, useful to compare the models. Accuracy in this context was the amount of entirely correct predictions (when the labels were rounded to 0 or 1) out of all predictions made.

For the 'prediction' and 'actual' MxN matrices (L^{pred} and L^{actual}) where M is the amount of data in a batch and N is the amount of floating point labels representing the classes of that data. The following formulae is how this paper defined this accuracy.

$$Acc_{AON} = \frac{N_{correct}}{N_{total}}$$

$$N_{correct} = \Sigma_{i=0}^{N_{total}} \left[\ \Pi_{j=0}^{n} \left(\ nint(L_{ij}^{pred}) \ \overline{\oplus} \ L_{ij}^{actual} \right) \ \right]$$

where: N_{total} is the amount of audio data per batch, $N_{correct}$ is the amount of correctly predicted data per batch, n is the number of labels for a given piece of data, *nint* represents a rounding function, L_{ij}^{pred} is the prediction for the i^{th} piece of audio's, j^{th} label, L_{ij}^{actual} is the actual i^{th} piece of audio's j^{th} label and $\overline{\oplus}$ represents the XNOR binary operation. Note: $int(L_{ij}^{pred})$ and L_{ij}^{actual} are either 0 or 1, since L_{ij}^{actual} is either 0 or 1 by definition and $L_{ij}^{pred} \in [0,1]$, which is the range of the Softmax and Sigmoid functions on the final layer.

This was used to compare the overall accuracy of one-hot encoded models and multi-hot encoded models since label accuracy is measured differently for these problems. For instance, categorical accuracy was used to measure accuracy for one-hot problems and binary accuracy was used to measure the accuracy of multi-hot problems. The following formulae, represent the categorical and binary accuracy metrics based on the Keras metrics source-code (Keras 2019).

For categorical accuracy:

$$Acc_c = \frac{\Sigma_{i=0}^{N_{total}} (matchedLabel_i)}{N_{total}}$$

where

$$matchedLabel_i = \begin{cases} 1, & \text{if } \ j = k \ \text{ where } j \text{ and } k \text{ are the indexes for labels} \\ & L_{ij}^{pred} \ \text{ and } \ L_{ik}^{actual} \ \text{ which are the greatest labels} \\ & \text{in their label matrices respectively (for the } i^{\text{th}} \text{ data).} \\ 0, & \text{otherwise, if } \ j \neq k \end{cases}$$

For binary accuracy:

$$Acc_b = \frac{\Sigma_{i=0}^{N_{total}} (correctLabel_i)}{N_{total}}$$

where

$$correctLabel_i = \frac{\Sigma_{j=0}^{n} \left(\ nint(L_{ij}^{pred}) \ \overline{\oplus} \ L_{ij}^{actual} \ \right)}{n}$$

Note: binary accuracy is similar to the AON accuracy except an average of correct labels is taken rather than the product

To show how they differ, given the one-hot scenario with 2 pieces of data and 4 labels:

$$pred = \begin{bmatrix} 0.7 & 0.2 & 0.05 & 0.05 \\ 0.3 & 0.05 & 0.4 & 0.25 \end{bmatrix}, \quad actual = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

$$matchedLabel = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \therefore \quad Acc_c = 0.5$$

$$correctLabel = \begin{bmatrix} 4 \\ 3 \end{bmatrix} \quad \therefore \quad Acc_b = 0.875$$

$$N_{correct} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \therefore \quad Acc_{AON} = 0.5$$

And given the multi-hot scenario with 2 pieces of data and 4 labels:

$$pred = \begin{bmatrix} 0.2 & 0.8 & 0.9 & 0.05 \\ 0.1 & 0.4 & 0.5 & 0.0 \end{bmatrix}, \quad actual = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$matchedLabel = \begin{bmatrix} ? \\ 1 \end{bmatrix}$$

since both L_{00}^{actual} and L_{10}^{actual} are equal and are the greatest labels in their row, Acc_c is undefined.

$$correctLabel = \begin{bmatrix} 2 \\ 4 \end{bmatrix} \quad \therefore \quad Acc_b = 0.75$$

$$N_{correct} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad \therefore \quad Acc_{AON} = 0.5$$

3.4.2 Confusion matrices

A confusion matrix is a square matrix of the one-hot encoded predictions against the one-hot encoded actual values (Powers 2007). For the multi-hot encoded problems, these were converted into one-hot encodings; however as the multi-hot models would occasionally predict infeasible kit combinations (that couldn't be one-hot encoded), these didn't appear in the confusion matrix and the number of these 'unclassified' predictions was noted.

3.4.3 Summary

Categorical accuracy was used as the label accuracy for one-hot problems as it is closer to class accuracy than binary accuracy for these problems but binary accuracy shall be used for multi-hot problems since categorical accuracy only works when in each 'actual' matrix of labels there is one label that is greater than the others. The AON accuracy works the same irrespective of whether one-hot or multi-hot encoding is used because it measures the accuracy of predicting entirely correct classes instead of the labels.

Each metric was measured with the loss during training per epoch for training data and validation data. Final values these metrics for the test data was calculated after 10 epochs of training and validation data.

For a given model we can observe underfitting when the training loss is significantly greater than the validation loss ($Loss_t > Loss_v$ indicates underfitting). Similarly, we can

observe overfitting when the training loss is significantly less than the validation loss ($Loss_t < Loss_v$ indicates overfitting). Optimally, these losses would be similar, showing that the model is generalising (Stallard and Taylor 1999).

Finally, confusion matrices were made using test data, for each configuration of each model, to observe distributions of where the model was correct or incorrect which is important in noticing what the common mistakes were and could provide insight into potential causes.

4 Analysis

4.1 Training

The different optimisation algorithms were tested prior to testing all of the models and configurations. It was found that Adam was both more efficient at converging towards a minimum value of the loss function per epoch (see Appendix C.1.1) and didn't affect computing time adversely in comparison to the other optimisation algorithms (see Appendix C.1.2).

The training times of each epoch for every model with the Adam optimiser was also recorded as well as the total time taken to complete 10 epochs. The totals on average were: Model A $\approx$ 266 minutes, Model B $\approx$ 202 minutes and Model C $\approx$ 220 minutes (see Appendix C.2). This suggests that the difference between the processing speed of the models is negligible since Model A's average was brought up by a single training session which took 426 minutes to complete. When ignoring this outlier, the average of Model A becomes $\approx$ 213 minutes.

For each model, the multi-hot encoded KLD loss configuration got stuck between epochs 1 and 2 during training at a slightly negative loss value and didn't learn.

4.1.1 Model A

As we can see from the loss and accuracy graphs for Model A in Appendix C.3, both of the one-hot encoded configurations were trained successfully and so was the multi-hot encoded configuration with BCE loss, however, the multi-hot encoded configuration with KLD loss didn't train.

When comparing the one-hot encoded configurations, BCE and KLD performed very similarly. With both losses, the training loss and validation loss converge by the 10^{th} epoch. Also, the accuracy curves of both are very similar, however, KLD was slightly more accurate than BCE throughout. KLD was quicker than BCE at minimising the loss for the first 5 epochs because the loss gradients were steeper, however with KLD the magnitude of the gradient is always steadily decreasing, but with BCE after the 4^{th} epoch the loss decreases at a steady rate and it minimises at a steeper gradient than KLD towards the end; catching it up in accuracy.

With multi-hot encoded BCE loss, the training and validation loss values don't converge, remaining a similar distance apart throughout the 10 epochs of training. Strangely, when KLD is used as a metric for BCE loss, the validation loss curve zig-zags, which coincides with a similar zig-zag on the AON validation accuracy curve.

4.1.2 Model B

For Model B (see Appendix C.4), again both of the one-hot encoded configurations were trained successfully, while only the multi-hot encoded configuration with BCE loss trained and KLD loss didn't train.

When comparing the one-hot encoded configurations, BCE and KLD again performed very similarly. With both losses, the training loss and validation loss converge by the 8^{th}-9^{th} epoch. Looking at the KLD loss for each by the 10^{th} epoch the training and validation loss looks like it could be diverging with validation being greater than training loss.

Again, the accuracy curves of both are very similar, however, KLD was slightly more accurate than BCE throughout. Unlike Model A, the loss gradients start to plateaux for both configurations so it is likely that Model B won't get much more accurate past 10 epochs.

With multi-hot encoded BCE loss, again the training and validation loss values don't converge, remaining a similar distance apart throughout the 10 epochs of training. The values of the loss curve for multi-hot encoding are significantly greater than those of both the one-hot encoded configurations.

4.1.3 Model C

For Model C (see Appendix C.5), when it was trained using linear magnitude spectrogram data the models didn't work. Either classifying all of the data to one class or leaving all of the data unclassified, so was very inaccurate.

When logarithmic magnitude spectrogram data was used only the configurations with BCE loss worked. However, as before with Models A and B, the one-hot encoded configuration minimised the loss to much lower values than the multi-hot configuration and the training and validation loss values were very close together with one-hot and further apart with multi-hot. With one-hot encoding the training and validation loss values meet by about the 7^{th} epoch, then they diverge, but the validation loss still decreases. We can also see this with the accuracy graphs for the one-hot encoded BCE loss configuration as well.

With logarithmic magnitude, both of the KLD loss configurations got stuck by the 2^{nd} epoch and didn't learn.

4.2 Evaluation

The evaluation of the models was done after each model and its configuration was trained for 10 epochs. The evaluation was done with a different 'testing' data set to the data sets used for training and validation. Metrics, amount of unclassified data and confusion matrices were produced during the model evaluations.

4.2.1 General observations

The confusion matrices show either a strong positive correlation for results to fall close to the identity matrix, where class predictions by the models are very accurate to the true class values or a vertical bar which shows that models guessed the same class for each true class and didn't learn. On less accurate models, the confusion matrices show a significant tendency to underestimate the number of classes in the data, as a fern-like pattern, made of vertical and diagonal lines, can be observed along the left side of the confusion matrix.

Whenever KLD loss was used with multi-hot encoding, the loss value would always get stuck at a value of -1.2910 x 10^{-5}. The resulting output would always have all of the labels set to 1.0 ('on'), which isn't a valid combination of classes. For this reason, there are no confusion matrices for these configurations since a guess has to be a valid combination to appear in the matrix. To account for this, it was recorded for each configuration how many times it made an 'unclassified' prediction. A prediction was unclassified when either: no labels were greater than or equal to 50% activation or when an invalid set of labels were over 50% activation.

4.2.2 Model A

Results

	Loss functions		Accuracy metrics			Unclassified
	BCE	KLD	Acc_c	Acc_b	Acc_{AON}	
One-hot & BCE	0.0163	0.9902	0.7750	-	0.6862	0.2569
One-hot & KLD	0.0161	0.9706	0.7765	-	**0.6930**	0.2411
Multi-hot & BCE	0.1699	0.9686	-	0.9366	0.5253	0.1977
Multi-hot & KLD	12.7702	0.0000	-	0.1990	0.0000	1.0000
Average	3.2431	0.7323	0.7758	0.5678	0.4761	0.4239

Figure 13: The results of Model A for the various configurations as a decimal percentage of the test data set.

Model A gave very similar accuracy results for both of the loss functions when one-hot encoding was used, but KLD was slightly more accurate (0.68%) than BCE. When multi-hot encoding was used, only BCE loss was successful at learning; with KLD having 0% (AON) class accuracy. Also, BCE with multi-hot encoding was significantly lower in accuracy (-16.8%) than BCE with one-hot encoding. Using one-hot encoding with KLD as the loss function gave the best AON accuracy for Model A.

Prediction distribution

By looking at the confusion matrices that Model A produced, it is clear that the model has a tendency to underestimate the number of classes in the data. This is particularly clear looking at Model A with multi-hot encoding and BCE loss (see Appendix D.1). For instance with data containing the 'normal hi-hat' class with another class: the distribution of predictions were mainly split between including and excluding the normal hi-hat class, with a small amount excluding the accompanying class and predicting a solo 'normal hi-hat'; this can be seen more generally to some degree for most of the data with more than one class.

Generally a less common mistake was overestimating the number of classes, but this occurred mainly with the solo tom drum classes and the solo 'bass drum' class. For the multi-hot encoded BCE loss configuration overestimation also happened where the model would add a 'high tom', 'snare' or 'crash' where there wasn't one. This tended to happen for the 'high tom' and 'snare' when the other was played and happened for the 'crash' when another cymbal was played.

Mistaking one class for another class was quite rare with one-hot encoding, although the place this happened the most was with the different hi-hat and ride cymbal techniques; particularly with mistaking a 'normal ride' hit for a 'ride bell' hit at 11% for one-hot encoding with KLD loss. This happened to a lesser extent than first expected with the different tom drums (2-5%). With multi-hot encoding, there were many false positive predictions with the classes containing a 'crash', which can be seen by the vertical lines on columns featuring a crash.

Overall, for the classified data from the configurations of Model A, one-hot encoding and BCE loss gave the most consistent looking confusion matrix with one-hot encoding and

KLD loss showing a slight tendency to underestimate the amount of classes and predicting incorrect techniques. Multi-hot encoding and BCE loss showed the most mistakes in its predictions however, made the more classified predictions than both of the one-hot configurations. Multi-hot encoding and BCE loss made large amounts of underestimation mistakes, had a tendency to overestimate as well and predicted incorrect classes with similar timbre too.

4.2.3 Model B

Results

	Loss functions		Accuracy metrics			Unclassified
	BCE	KLD	Acc_c	Acc_b	Acc_{AON}	
One-hot & BCE	0.0108	0.6342	0.8508	-	0.8153	0.1274
One-hot & KLD	0.0098	0.5697	0.8691	-	**0.8380**	0.1238
Multi-hot & BCE	0.1692	1.1614	-	0.9353	0.5067	0.2189
Multi-hot & KLD	12.7702	0.0000	-	0.1990	0.0000	1.0000
Average	3.2400	0.5913	0.8600	0.5672	0.5400	0.3675

Figure 14: The results of Model B for the various configurations as a decimal percentage of the test data set.

Model B, like Model A, gave very similar accuracy results for both of the loss functions, when one-hot encoding was used. Again KLD was slightly more accurate (2.27%) than BCE. When multi-hot encoding was used, only BCE loss was successful at learning; with KLD having 0% (AON) class accuracy. Also, BCE with multi-hot encoding was significantly lower in accuracy (-30.8%) than BCE with one-hot encoding.

Prediction distribution

From the confusion matrices obtained from Model B (see Appendix D.2), the same behaviour can be observed from Model A but to a lesser extent. One-hot encoded configurations were correct for the majority of the time. The classes that these configurations struggled with identifying the most were the tom drums and the different techniques of ride cymbal ('normal ride' and 'ride bell'). Model B did a lot less overestimation of the amount of classes in some data, however the multi-hot encoding with BCE loss configuration still heavily underestimated the amount of classes and did this more so than Model A. For instance it actually misidentified 'high tom' and 'mid tom' as solo 'mid tom' at a rate of 63%. The multi-hot encoding with BCE loss configuration was more accurate in predicting solo classes than one-hot encoding with BCE loss, but it can be seen that the multi-hot configuration had much larger amounts of false positives for these classes; especially with solo 'bass drum' false positives.

Overall, for the classified data from the configurations of Model B, one-hot encoding and BCE loss gave the most consistent confusion matrix with one-hot encoding and KLD loss showing a very slight tendency to underestimate the amount of classes and predicting incorrect techniques although this is only significant with classes containing a 'crash'. Multi-hot encoding and BCE loss showed the most mistakes in its predictions. Multi-hot

encoding and BCE loss made large amounts of underestimation mistakes, had a tendency to overestimate as well and predicted incorrect classes with similar timbre too.

4.2.4 Model C

Results

		Loss functions		Accuracy metrics			Unclassified
		BCE	KLD	Acc_c	Acc_b	Acc_{AON}	
Linear spectrogram	One-hot & BCE	0.0602	4.2490	0.0334	-	0.0000	1.0000
	One-hot & KLD	0.3660	16.0070	0.0069	-	0.0069	0.0000
	Multi-hot & BCE	5.6115	20.1146	-	0.6494	0.0093	0.0000
	Multi-hot & KLD	12.7702	0.0000	-	0.1990	0.0000	1.0000
	Average	4.7020	10.0926	0.0201	0.4242	0.0041	0.5000
Log spectrogram	One-hot & BCE	0.0175	1.0263	0.7154	-	**0.6264**	0.2906
	One-hot & KLD	0.3125	15.9224	0.0094	-	0.0064	0.0000
	Multi-hot & BCE	0.1909	1.1920	-	0.9277	0.4691	0.1067
	Multi-hot & KLD	12.7702	0.0000	-	0.1990	0.0000	1.0000
	Average	3.3228	4.5352	0.4047	0.5634	0.2755	0.3493

Figure 15: The results of Model C for the various configurations as a decimal percentage of the test data set.

With Model C, linear magnitude spectrograms performed poorly with none of the configurations reaching 1% (AON) class accuracy. Logarithmic magnitude spectrograms with KLD didn't learn, irrespective of the encoding used, but when using BCE loss they did learn. Again one-hot encoding did significantly better than multi-hot encoding with a difference of 15.7%.

Prediction distribution

With linear magnitude spectrograms the confusion matrices show why these were so inaccurate (see Appendix D.3), because they selected the same classes for all of the data. For one-hot encoding with KLD loss all of the predictions were for 'bass drum' with 'open hi-hat' and with 'high tom', while for multi-hot encoding with BCE loss all of the predictions were for 'bass drum' with 'normal hi-hat' and 'normal ride'. Both one-hot encoding with BCE loss and multi-hot encoding with KLD loss left 100% of test data as unclassified, producing no confusion matrices.

With logarithmic magnitude spectrograms the confusion matrices show that for BCE loss configurations the models tend to underestimate as well as slightly underestimate the number of classes as before with Models A and B. An interesting new feature of Model C's confusion matrices for BCE loss are that diagonal lines appear either side of the middle diagonal, suggesting that the model had some trouble in particular with deciding the presence of the 'bass drum' class in the data; sometimes including it where it wasn't there or excluding it when it was.

Overall, the most consistent configuration of Model C was one-hot encoding with BCE loss on input data of logarithmic magnitude, however this had particular difficulty differentiating between different drum classes (for example, mid-tom was predicted as low-tom incorrectly 19% of the time) and was also prone to underestimating the number of classes;

in particular with the bass drum. Multi-hot encoding with BCE loss on input data of logarithmic magnitude was the other model which learnt, but was very inaccurate especially with identifying multi-classed data as it heavily underestimated the amount of classes in the data; this caused very low levels of accuracy for hits with 3 classes. Configurations with input of linear magnitude didn't learn at all either having 100% of data unclassified or predicting the same class.

5 Conclusion

5.1 Comparison of models

The best model was Model B because it learned at a faster rate than the others and reached a higher level of accuracy over 10 epochs. In particular, the one-hot encoded variant performed the best, reaching the highest accuracy (see section 4.2.3). Model B appears to have generalised by the 8^{th} epoch for KLD loss and the 10^{th} epoch for BCE loss since the training and validation loss curves meet at this point (see Appendix C.4). Training this model much further than 10 epochs, could risk the model overfitting.

5.2 Comparison of input types

For the models that were tested. Model C didn't learn from linear spectrogram data, however, Model C did learn with logarithmic spectrogram data, the best configuration was 62.64% accurate at predicting the correct drum kit classes. Models using amplitude data over time performed better than spectrogram data getting 69.30% class accuracy for Model A and 83.80% class accuracy for Model B. More models need to be tested in order to make a fair comparison as to whether 1D amplitude data is better than 2D spectrogram data for training drum kit timbre CNN models.

Given more time I would have made another model for 2D spectrogram data, as it could be the model architecture that was reducing the accuracy instead of the format of the input data. However, this is hard to test because architectures need to change to accommodate for different formats of input data.

5.3 Comparison of encoding types

Over the models and losses tested, one-hot encoding consistently outperformed multi-hot encoding by significant margins of accuracy. This was expected, since only 87 combinations of drum kit hits were valid combinations and with multi-hot encoding had the capability to encode 1024 (2^{10}) combinations (937 of these were infeasible), which to the neural network started with equal probability. This undoubtedly made training take longer. An improvement would have been to use a smaller amount of bits of information in the output, but 87 doesn't partition into 2's to make an integer, so the closest we could get would be 7 bits and still have 41 infeasible combinations. Surprisingly, multi-hot encoding with BCE loss for models A and C left less data unclassified than the one-hot configurations despite the large amount of infeasible encodings, but this wasn't the case with Model B.

5.4 Comparison of loss functions

With one-hot encoding, KLD had slightly better accuracy compared to BCE, across all of the models. However, with multi-hot encoding, KLD gets stuck and doesn't learn at all whilst BCE loss configurations do learn. It is unclear what causes KLD to get stuck at a negative value, since as well as the encoding changing, the final activation layer changes which could impact the loss function. With each model, multi-hot with a KLD loss always set all of the labels to 1.0 ('on'), so it consistently had a class accuracy of 0.0%.

5.5 Final model

The final model (v6) that was improved upon from Model B with one-hot encoding is shown in Figure 16.

	Layers		Activation	Matrix shape	Amount
Initial down-sampling layers	CC 7/8 (s = 3)		-	$(B, 12000, 1)$	x1
	CC 7/16 (s = 2)		Leaky ReLU (g = 0.05)	$(B, 4000, 8)$	
	CC 5/32 (s = 2)		Leaky ReLU (g = 0.05)	$(B, 2000, 16)$	
	CC 3/64 (s = 2)		Leaky ReLU (g = 0.05)	$(B, 1000, 32)$	
	Dropout layer (rate = 0.1)		-	$(B, 500, 64)$	
Stacked dilated CC layers	Skip	CC 1/1	-	$(B, 500, 64)$	x4
		Dilated CC 3/16 (d = 2)	-	$(B, 500, 1)$	
		Dilated CC 3/64 (d = 2)	Leaky ReLU (g = 0.05)	$(B, 500, 16)$	
	Matrix addition		-	$(B, 500, 64)$	
	Dropout layer (rates: 0.15, 0.20, 0.25)		-	$(B, 500, 64)$	
Flatten and dense layers	Flattened layer		-	$(B, 500, 64)$	x1
	Dense layer		Softmax or Sigmoid	$(B, 32000)$	
	Output		-	$(B, 87)$	

Figure 16: A tabular representation of the structure of the final model, where: CC layers denoted with 'x/y' means x kernel size by y filters, 's' is the stride, 'rate' is the probability of a dropped connection, 'd' is the dilation rate and 'g' is the gradient of the negative part of the Leaky ReLU activation function. The matrix shape (a, b, c) shows the batch size (a), sample space size (b) and the feature space size (c); after the flattened layer, the sample space and feature space become the same dimension ($b \times c$). Note: for CCs where the stride and dilation rate are not specified, the stride is 1 and the dilation rate is 1 (no dilation).

From the prior comparisons drawn, a final model was made by experimenting with Model B (with one-hot encoding) on a larger input data set (see Figure 17). This model was initially trained with KLD loss since this performed best out of the configurations of Model B, but this loss didn't scale well when Model B was made larger, so BCE loss was used instead as it performed similarly to KLD loss (see section 4.1.2). The final model was slightly deeper having 4 repeated sections of stacked dilated CC layers instead of 3, it used twice the amount of filter space (64 instead of 32) and the Leaky ReLU gradient has reduced to 0.05 from 0.3. Using a batch size of 100, version 6 of the final model got to 92.4% accuracy with the larger data set after 10 epochs. When the original Model B was trained and tested on the larger data set with a batch size of 100, it scored 85.8%; so the final model was a 6.6% improvement upon Model B under the same conditions.

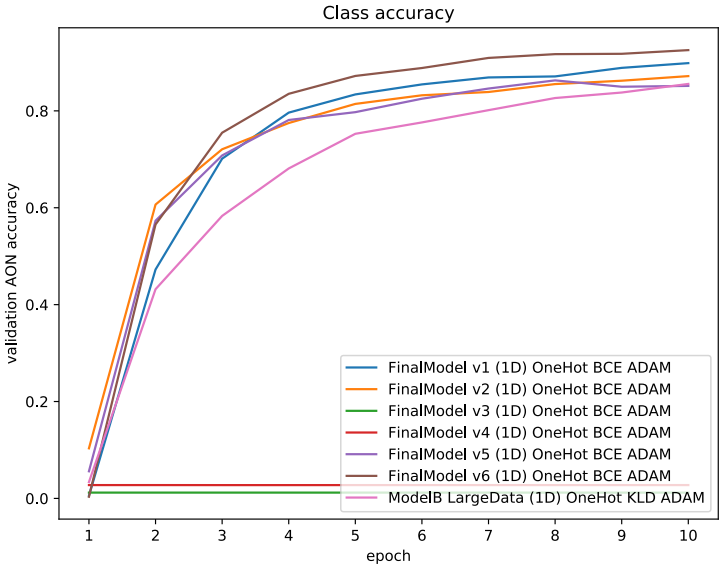


Figure 17: A comparison of the final models in class accuracy over 10 epochs.

5.6 Summary

Following the methodology, 3 out of the 4 objectives (see section 1.2) were met: 3 neural network models were designed and tested using synthetic multi-classed data from processed, isolated hits; drum kit timbre was shown to be a trainable feature across all of the classes; and a model was trained to a high degree of accuracy with the synthetic data.

The best performing model was Model B, the one-hot with KLD loss configuration reaching 83.8% (AON) class accuracy when evaluated using the original testing data set. Using a larger batch size and a larger data set, Model B was re-trained and got to 85.8% accuracy. Then, variations of Model B which tweaked depth, feature space and Leaky ReLU gradient were trained and tested, improving the accuracy to 92.4%.

This was close to the accuracy of the other drum kit classification model from the literature review (see section 2.3) which got 99% accuracy, but this model classified 10 classes containing audio of similar timbres compared to the model in the literature, which classified 3 classes which had distinct timbres.

Unfortunately, there was not enough time to apply this final model to real drum beats, which was the last objective. So it is not known how accurate these models are with real data.

5.7 Future work

With more time, a 4^{th} model could have been made using 2D spectrogram data with a structure using inception modules; like in Model A. This was because only one model was made for 2D spectrogram data while 2 models were made for 1D amplitude data and this model design might have performed better than Model C, since the original NN which used the inception module design (GoogLeNet), was designed for 2D data (Oord et al. 2016).

In addition to varying the output encoding and the loss function, the width and depth of the models could have been varied to examine their effect on training and overall accuracy. However, these were kept constant until the final model so that there weren't too many factors; as these combinations of factors increases the amount of configurations per model and each model takes a lot of time to train per configuration.

Since, there wasn't enough time to test this neural network model on real drum beats (it was only trained and tested on synthetic multi-classed hits), it isn't known how well it would perform on real data. This would be an interesting area to look into because even if the performance is slightly lower, the trained models can be adjusted to a higher accuracy, with less 'real' data than is necessary for training models from scratch by using transfer learning (Asadi and Huber 2007).

6 Bibliography

References

Asadi, Mehran and Manfred Huber (2007). "Effective control knowledge transfer through learning skill and representation hierarchies". In: *IJCAI International Joint Conference on Artificial Intelligence*, pp. 2054–2059.

Bello, Juan Pablo et al. (2005). "A tutorial on onset detection in music signals". In: *IEEE Transactions on Speech and Audio Processing* 13.5, pp. 1035–1046. DOI: 10.1109/TSA.2005.851998.

Benetos, Emmanouil et al. (2013). "Automatic music transcription: Challenges and future directions". In: *Journal of Intelligent Information Systems* 41.3, pp. 407–434. DOI: 10.1007/s10844-013-0258-3.

Cawsey, Alison (1998). *The essence of artificial intelligence*. Prentice Hall Europe, ix, 190 p. DOI: 10.1037/h0072647. URL: http://www.loc.gov/catdir/toc/fy034/97011460.html.

Chawla, Nitesh V et al. (2002). *SMOTE: Synthetic Minority Over-sampling Technique*. Tech. rep., pp. 321–357.

Glorot, Xavier and Yoshua Bengio (2010). "Understanding the difficulty of training deep feedforward neural networks". In: 9, pp. 249–256.

He, Kaiming et al. (2015a). "Deep Residual Learning for Image Recognition". URL: http://arxiv.org/abs/1512.03385.

— (2015b). "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: arXiv: 1502.01852. URL: http://arxiv.org/abs/1502.01852.

Hinton, Geoffrey E et al. (2014). *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*. Tech. rep. University of Toronto. URL: https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf.

Hubel, D. H. and T. N. Wiesel (1959). *Receptive fields of single neurones in the cat's striate cortex*. Tech. rep. 3, pp. 574–591. DOI: 10.1113/jphysiol.1959.sp006308. URL: http://doi.wiley.com/10.1113/jphysiol.1959.sp006308.

Ioffe, Sersey and Christian Szegedy (2015). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". URL: https://arxiv.org/abs/1502.03167v3.

Keras (2019). *Keras metrics source-code*. URL: https://github.com/keras-team/keras.

Krizhevsky, Alex, Geoffrey E Hinton, and Ilya Sutskever (2012). *ImageNet Classification with Deep Convolutional Neural Networks*. Tech. rep. Univeristy of Toronto, pp. 1–9. DOI: http://dx.doi.org/10.1016/j.protcy.2014.09.007. arXiv: 1102.0183. URL: https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

Lecun, Yann et al. (1998). *Gradient-Based Learning Applied to Document Recognition*. Tech. rep. IEEE. DOI: 10.1109/5.726791. URL: https://ieeexplore.ieee.org/document/726791.

LeCunn, Yann (2010). *THE MNIST DATABASE of handwritten digits*. DOI: 10.1109/70.34763. URL: http://yann.lecun.com/exdb/mnist/.

Librosa (2019). *Librosa library*. URL: https://github.com/librosa/librosa.

Lunaverus (2019). *Music Transcription with Convolutional Neural Networks*. URL: https://www.lunaverus.com/cnn.

Maas, Andrew L, Awni Y Hannun, and Andrew Y Ng (2013). *Rectifier Nonlinearities Improve Neural Network Acoustic Models*. Tech. rep.

MacKay, David J. C. (2013). *Information theory, inference, and learning algorithms*. Vol. 41. 10. DOI: 10.5860/choice.41-5949. URL: https://www.inference.org.uk/itprnn/book.pdf.

Maxwell, Andrew et al. (2017). *Deep learning architectures for multi-label classification of intelligent health risk prediction*. BMC Bioinformatics. DOI: 10.1186/s12859-017-1898-z.

McCullock, Warren and Walter Pitts (1943). "A logical calculus of the ideas immanent in nervous activity". In: *Bulletin of mathematical biology* 52.1-2, pp. 99–115. DOI: 10.1007/BF02459570. URL: https://www.researchgate.net/publication/20969919_A_logical_calculus_of_the_ideas_immanent_in_nervous_activity_1943.

Merriam-Webster (2019). *Merriam-Webster*. URL: https://www.merriam-webster.com/dictionary/waveform.

Müller, Meinard (2015). *Fundamentals of Music Processing*. Springer-Verlag. DOI: 10.1007/978-3-319-21945-5. URL: http://link.springer.com/10.1007/978-3-319-21945-5.

Nair, V, GE Hinton - Proceedings of the 27th international Conference, and Undefined 2010 (2010). *Rectified Linear Units Improve Restricted Boltzmann Machines*. Tech. rep. University of Toronto, p. 6421113. DOI: 10.1.1.165.6419. arXiv: 1111.6189v1. URL: https://www.cs.toronto.edu/~hinton/absps/reluICML.pdf.

Nielsen, Michael A (2018). *Neural Networks and Deep Learning*. URL: http://neuralnetworksanddeeplearning.com/chap6.html.

Nwankpa, Chigozie et al. (2018). "Activation Functions: Comparison of trends in Practice and Research for Deep Learning". In: arXiv: 1811.03378. URL: http://arxiv.org/abs/1811.03378.

Oord, Aaron van den et al. (2016). "WaveNet: A Generative Model for Raw Audio". London. URL: http://arxiv.org/abs/1609.03499.

Paulus, J and T Virtanen (2005). *Drum transcription with non-negative spectrogram factorisation*. Tech. rep. URL: http://www.cs.tut.fi/sgn/arg/paulus/eusipco05_paulus.pdf.

Paulus, Jouni (2010). *Signal processing methods for drum transcription and music structure analysis*. Tech. rep. DOI: DOI:10.1007/s11214-007-9186-2. URL: http://dspace.cc.tut.fi/dpub/handle/123456789/6445.

Powers, David M W (2007). *Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation*. Tech. rep. Adelaide: Flinders University of South Australia.

Raschka, Sebastian (2015). *Single-Layer Neural Networks and Gradient Descent*. URL: https://sebastianraschka.com/Articles/2015_singlelayer_neurons.html.

Rojas, Raul (1996). *Neural Networks - A Systematic Introduction - Backpropagation*. Springer-Verlag, pp. 152–184. DOI: 10.1109/78.127967. URL: https://page.mi.fu-berlin.de/rojas/neural/neuron.pdf.

Rosenblatt, F (1958). *THE PERCEPTRON: A PROBABILISTIC MODEL FOR INFORMATION STORAGE AND ORGANIZATION IN THE BRAIN 1*. Tech. rep. 6. Cornell Aeronautical Laboratory, pp. 19–27.

Russell, Stuart and Peter Norvig (1995). *Artificial Intelligence: A Modern Approach*. Vol. 9. 2, pp. 215–218. DOI: 10.1016/0925-2312(95)90020-9. URL: http://portal.acm.org/citation.cfm?id=773294.

SciPy (2019). *SciPy.org*. URL: https://www.scipy.org/.

Springenberg, Jost Tobias et al. (2014). "Striving for Simplicity: The All Convolutional Net". URL: http://arxiv.org/abs/1412.6806.

Stallard, Brian R and John G Taylor (1999). *Quantifying multivariate classification performance-the problem of overfitting*. Tech. rep. Albuquerque: Sandia National Laboratories. URL: https://www.researchgate.net/publication/253914692_Quantifying_multivariate_classification_performance_the_problem_of_overfitting.

Szegedy, Christian et al. (2015). "Going deeper with convolutions".

Wong, Sebastien C. et al. (2016). *Understanding Data Augmentation for Classification: When to Warp?* DOI: 10.1109/DICTA.2016.7797091. arXiv: arXiv:1609.08764v1. URL: https://arxiv.org/abs/1609.08764v2.

Yokoyama, Masao, Yoshiki Awahara, and Genki Yagawa (2016). "Relation between violin timbre and harmony overtone". In: *The Journal of the Acoustical Society of America* 140.4, pp. 3427–3427. DOI: 10.1121/1.4971030.

Figures

12019, ID (2017). *Kingfisher*. [Online, adapted in accordance with the Pixabay License; accessed on 23-01-2019]. URL: https://pixabay.com/en/kingfisher-bird-wildlife-macro-2046453/.

Hashmi, Syed Wamiq Ahmed (2013). *Drums schematic*. [Online, used in accordance with the Wikipedia Commons, creative commons license; accessed on 21-01-2019]. URL: https://upload.wikimedia.org/wikipedia/commons/5/57/Drums_schematic.svg.

7 Appendices

A Input data

A.1 Drum and cymbal classification

Referred to in section 3.1.3

In *settings.json* the type of hits, kit components and techniques used are described by the *label* associative array. The combinations of these types which are valid classes are described by the *label-hierarchy* associative array. Here is the section of this file where they appear:

```
"label": {
    "hit_label": ["beater", "stick"],
    "kit_label": [
      "bass_drum",
      "crash",
      "hi_hat",
      "high_tom",
      "mid_tom",
      "low_tom",
      "ride",
      "snare",
      "splash"
    ],
    "tech_label": ["bell", "normal", "open"]
},

"label-hierarchy": {
    "beater": { "bass_drum": ["normal"] },
    "stick": {
      "crash": ["normal"],
      "hi_hat": ["normal", "open"],
      "high_tom": ["normal"],
      "mid_tom": ["normal"],
      "low_tom": ["normal"],
      "ride": ["normal", "bell"],
      "snare": ["normal"]
    }
}
```

Later the splash cymbal wasn't used because there wasn't enough raw data for it. Hence it doesn't appear in *label-hierarchy* even though its still in *label*.

A.2 Drum kit combinations

`Referred to in section 3.1.3`

There are 87 different combinations of drum and cymbal that can be hit at once, since all drums and cymbals are hand operated apart from the bass drum and opening/closing hi-hat (not counting the hi-hat closing shut as a hit). The following list is how encodings were ordered:

Index	Drum kit classes		
1	bass-drum normal		
2	crash normal		
3	hi-hat normal		
4	hi-hat open		
5	high-tom normal		
6	low-tom normal		
7	mid-tom normal		
8	ride bell		
9	ride normal		
10	snare normal		
11	bass-drum normal	crash normal	
12	bass-drum normal	hi-hat normal	
13	bass-drum normal	hi-hat open	
14	bass-drum normal	high-tom normal	
15	bass-drum normal	low-tom normal	
16	bass-drum normal	mid-tom normal	
17	bass-drum normal	ride bell	
18	bass-drum normal	ride normal	
19	bass-drum normal	snare normal	
20	crash normal	hi-hat normal	
21	crash normal	hi-hat open	
22	crash normal	high-tom normal	
23	crash normal	low-tom normal	
24	crash normal	mid-tom normal	
25	crash normal	ride bell	
26	crash normal	ride normal	
27	crash normal	snare normal	
28	hi-hat normal	high-tom normal	
29	hi-hat normal	low-tom normal	
30	hi-hat normal	mid-tom normal	
31	hi-hat normal	ride bell	
32	hi-hat normal	ride normal	
33	hi-hat normal	snare normal	
34	hi-hat open	high-tom normal	
35	hi-hat open	low-tom normal	
36	hi-hat open	mid-tom normal	
37	hi-hat open	ride bell	
38	hi-hat open	ride normal	
39	hi-hat open	snare normal	
40	high-tom normal	low-tom normal	
41	high-tom normal	mid-tom normal	
42	high-tom normal	ride bell	
43	high-tom normal	ride normal	
44	high-tom normal	snare normal	

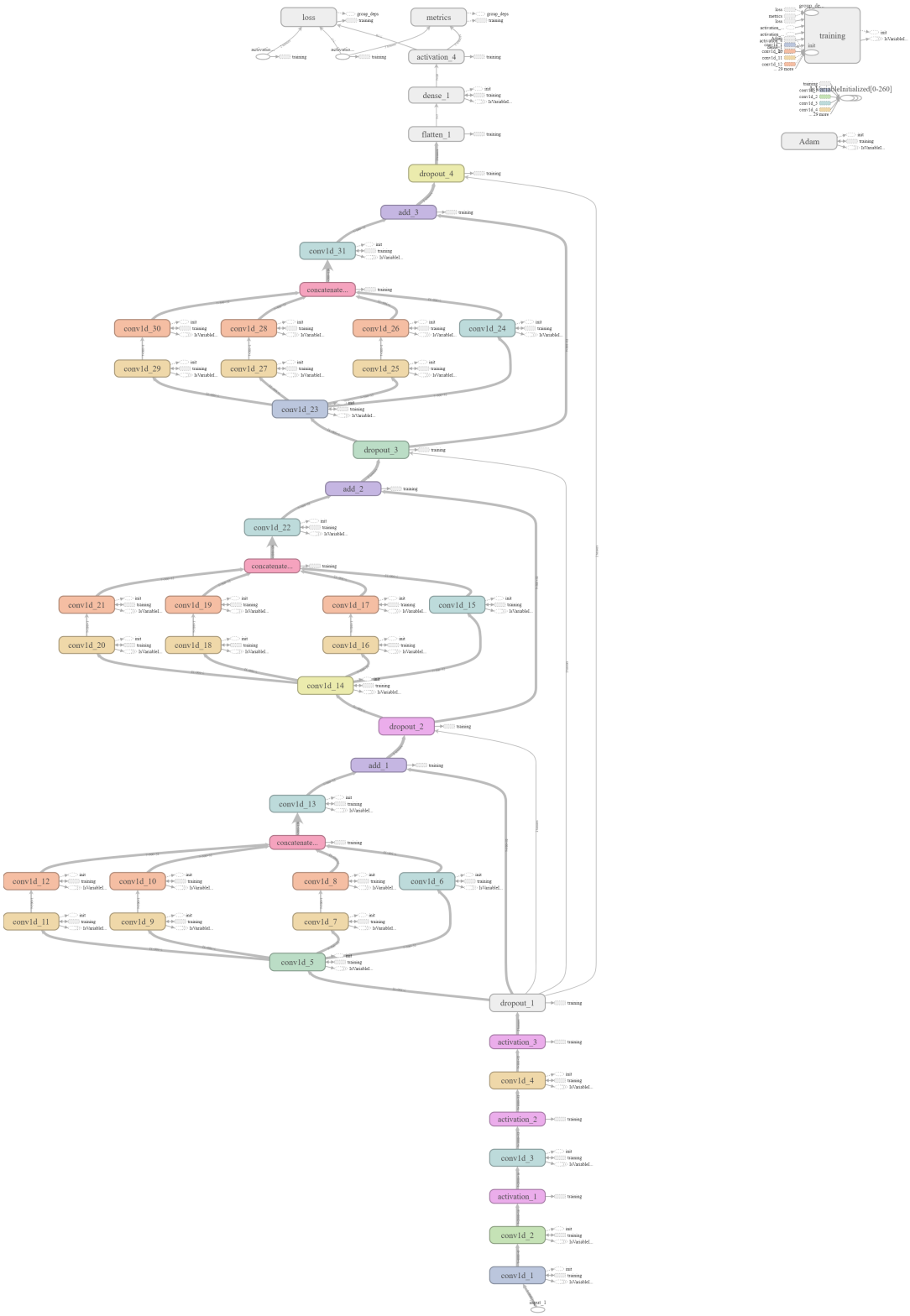
Index	Drum kit classes		
45	low-tom normal	mid-tom normal	
46	low-tom normal	ride bell	
47	low-tom normal	ride normal	
48	low-tom normal	snare normal	
49	mid-tom normal	ride bell	
50	mid-tom normal	ride normal	
51	mid-tom normal	snare normal	
52	ride bell	snare normal	
53	ride normal	snare normal	
54	bass-drum normal	crash normal	hi-hat normal
55	bass-drum normal	crash normal	hi-hat open
56	bass-drum normal	crash normal	high-tom normal
57	bass-drum normal	crash normal	low-tom normal
58	bass-drum normal	crash normal	mid-tom normal
59	bass-drum normal	crash normal	ride bell
60	bass-drum normal	crash normal	ride normal
61	bass-drum normal	crash normal	snare normal
62	bass-drum normal	hi-hat normal	high-tom normal
63	bass-drum normal	hi-hat normal	low-tom normal
64	bass-drum normal	hi-hat normal	mid-tom normal
65	bass-drum normal	hi-hat normal	ride bell
66	bass-drum normal	hi-hat normal	ride normal
67	bass-drum normal	hi-hat normal	snare normal
68	bass-drum normal	hi-hat open	high-tom normal
69	bass-drum normal	hi-hat open	low-tom normal
70	bass-drum normal	hi-hat open	mid-tom normal
71	bass-drum normal	hi-hat open	ride bell
72	bass-drum normal	hi-hat open	ride normal
73	bass-drum normal	hi-hat open	snare normal
74	bass-drum normal	high-tom normal	low-tom normal
75	bass-drum normal	high-tom normal	mid-tom normal
76	bass-drum normal	high-tom normal	ride bell
77	bass-drum normal	high-tom normal	ride normal
78	bass-drum normal	high-tom normal	snare normal
79	bass-drum normal	low-tom normal	mid-tom normal
80	bass-drum normal	low-tom normal	ride bell
81	bass-drum normal	low-tom normal	ride normal
82	bass-drum normal	low-tom normal	snare normal
83	bass-drum normal	mid-tom normal	ride bell
84	bass-drum normal	mid-tom normal	ride normal
85	bass-drum normal	mid-tom normal	snare normal
86	bass-drum normal	ride bell	snare normal
87	bass-drum normal	ride normal	snare normal

Valid drum kit combinations

B Model structure

B.1 Model A

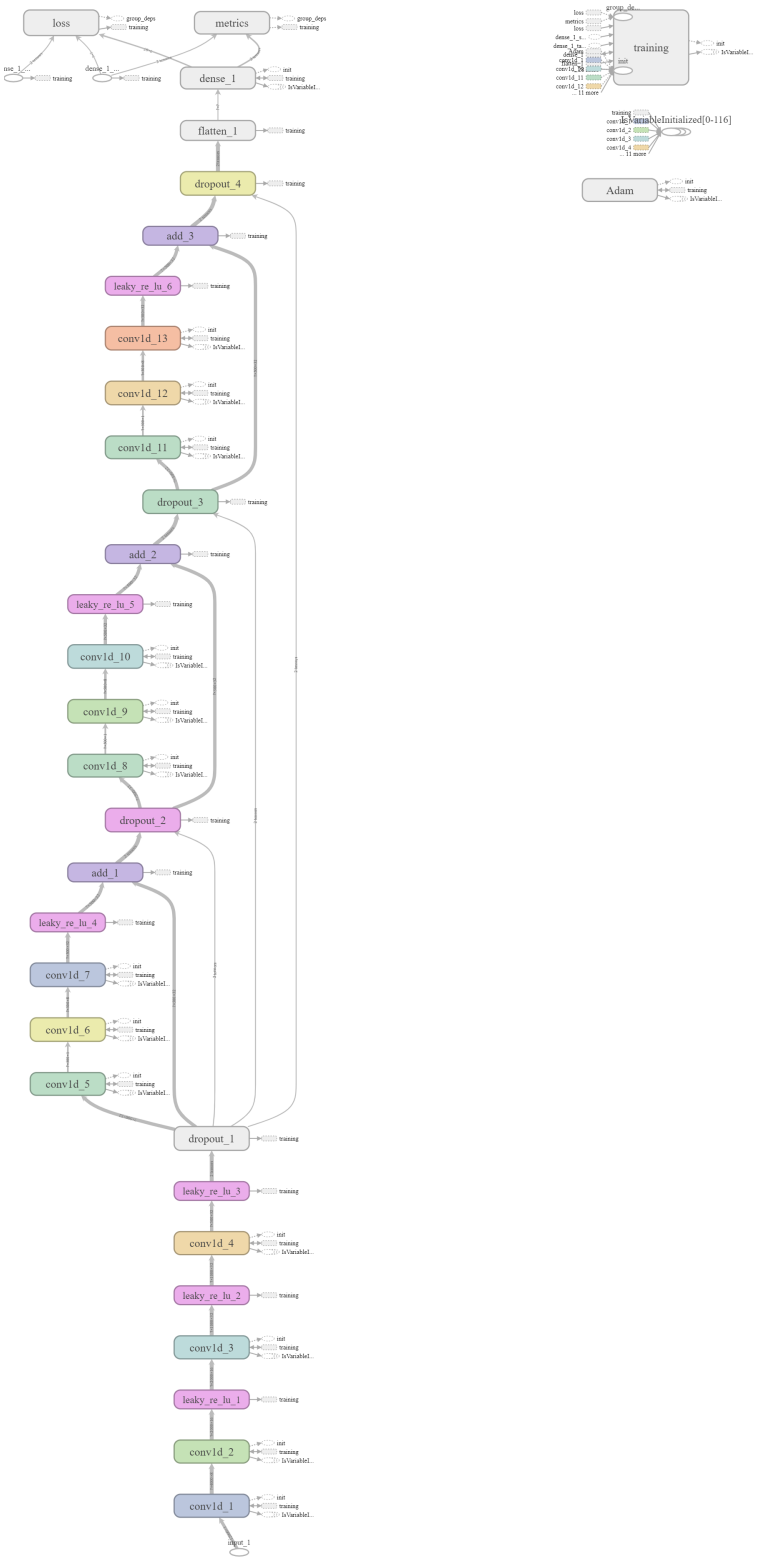
Referred to in section 3.2.1



TensorBoard diagram of Model A.

B.2 Model B

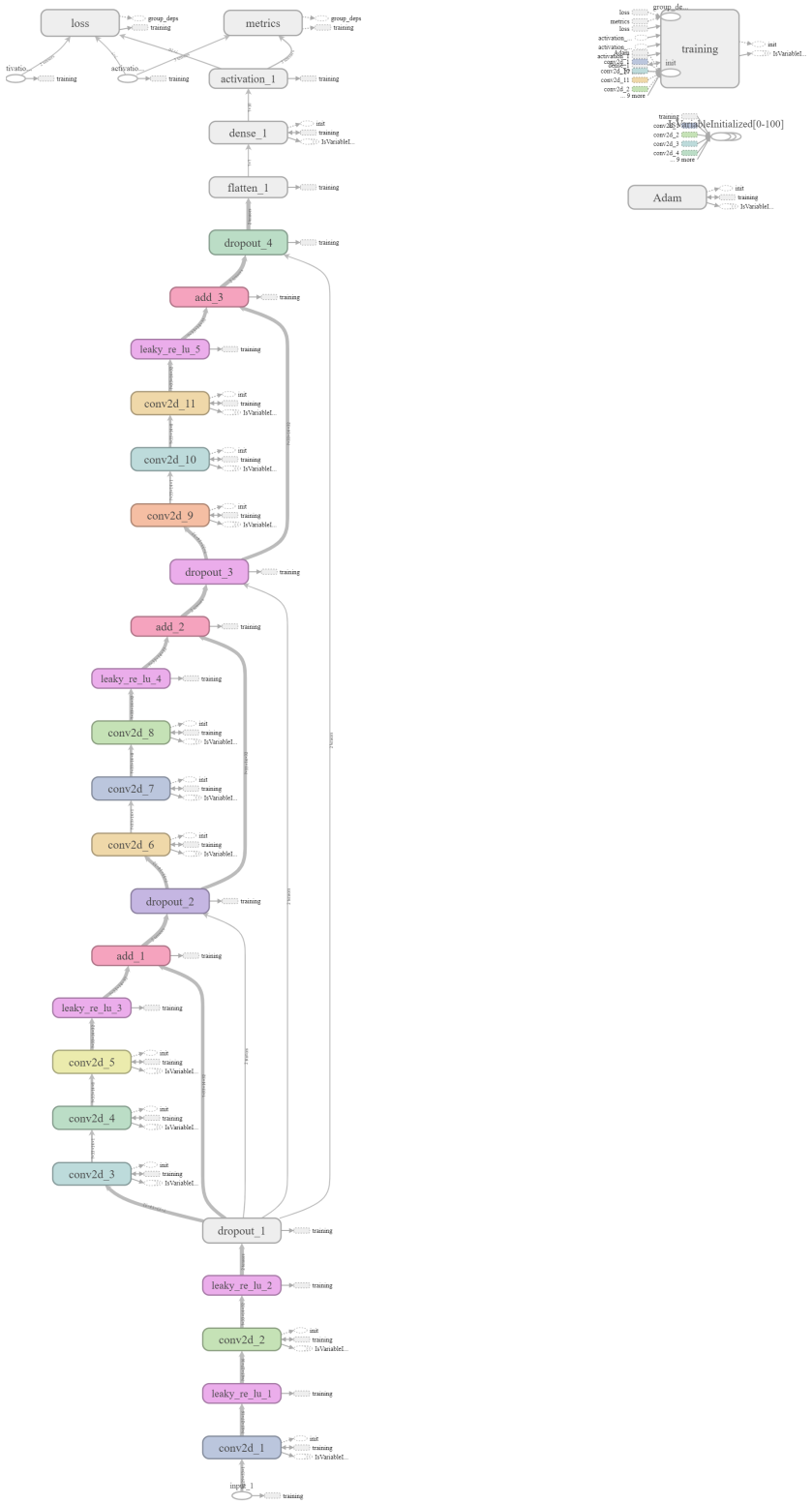
Referred to in section 3.2.2



TensorBoard diagram of Model B.

B.3 Model C

Referred to in section 3.2.3



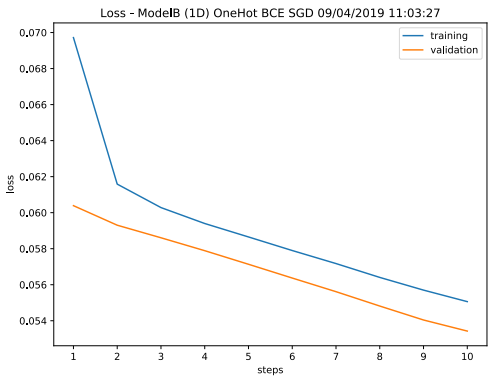
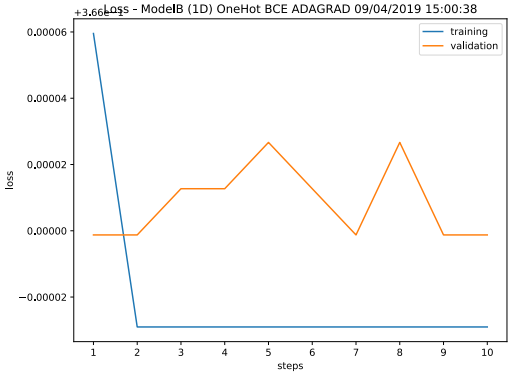
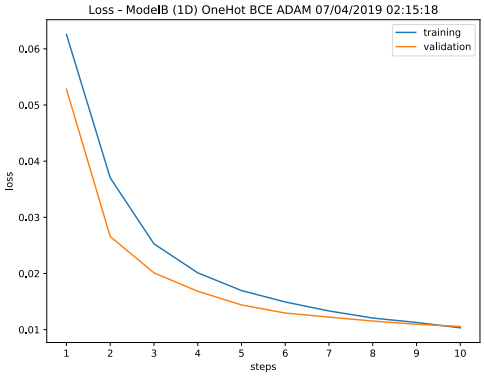
TensorBoard diagram of Model C.

C Training progress

C.1 Optimisation algorithm comparison

C.1.1 Loss during training

`Referred to in section 4.1`

Optimiser	Loss and validation loss during training
SGD	
ADAGRAD	
ADAM	

The loss values during training for Model B with one-hot encoding and BCE loss when using different optimiser algorithms. *Note: the vertical scale of the middle figure (for ADAGRAD) is +0.366.*

C.1.2 Training processing times

Referred to in section 4.1

Optimiser	Epoch										Total
	1	2	3	4	5	6	7	8	9	10	
ADAGRAD	24.0	23.0	23.1	25.1	21.4	24.3	26.9	22.3	22.2	24.8	237.0
ADAM	24.3	19.0	19.0	19.0	19.1	19.0	18.9	19.1	19.0	18.9	195.3
SGD	23.6	22.3	22.7	28.4	21.8	24.2	19.3	18.9	19.0	18.9	219.0

The duration of each epoch as well as total time taken for 10 epochs in minutes. For different optimisation algorithms on Model B with one-hot encoding and BCE loss.

C.2 All model processing times with ADAM optimiser

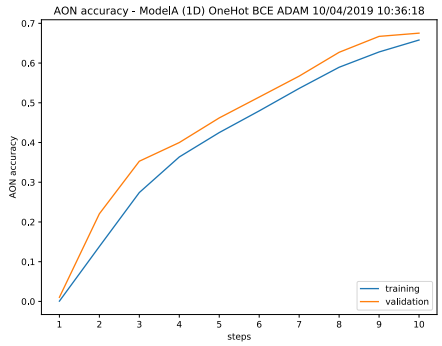
Referred to in section 4.1

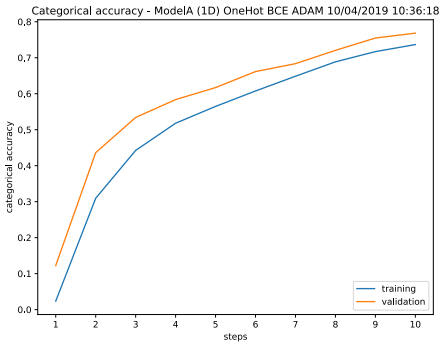
Model and configuration	Epoch										Total
	1	2	3	4	5	6	7	8	9	10	
ModelA (1D) MultiHot BCE	24.8	19.6	19.1	19.3	19.2	19.2	19.3	19.4	19.1	19.2	198.1
ModelA (1D) MultiHot KLD	25.7	20.2	19.5	19.4	19.3	21.8	22.7	25.9	19.2	19.2	212.8
ModelA (1D) OneHot BCE	35.6	44.4	42.7	46.6	43.1	43.3	42.2	42.5	42.2	43.5	426.0
ModelA (1D) OneHot KLD	24.0	24.5	26.5	34.6	20.2	19.4	19.5	19.3	19.5	19.5	227.0
ModelB (1D) MultiHot BCE	24.3	27.3	22.2	19.0	18.7	18.7	19.0	19.1	18.8	18.9	205.9
ModelB (1D) MultiHot KLD	25.8	22.6	20.2	20.7	20.2	20.2	20.4	20.2	20.1	20.2	210.6
ModelB (1D) OneHot BCE	24.3	19.0	19.0	19.0	19.1	19.0	18.9	19.1	19.0	18.9	195.3
ModelB (1D) OneHot KLD	24.7	19.2	18.8	19.0	19.0	18.9	19.0	19.1	19.3	19.3	196.4
ModelC (lin 2D) MultiHot BCE	24.3	18.8	19.0	19.0	19.0	19.1	19.0	19.0	19.3	23.9	200.3
ModelC (lin 2D) MultiHot KLD	24.8	24.1	29.4	32.5	22.7	19.2	19.1	19.0	19.0	19.2	229.0
ModelC (lin 2D) OneHot BCE	24.2	19.0	19.0	19.0	19.0	18.9	18.9	19.3	19.0	18.9	195.1
ModelC (lin 2D) OneHot KLD	23.4	18.8	19.0	19.1	23.6	24.7	24.9	25.5	22.6	25.4	227.0
ModelC (log 2D) MultiHot BCE	24.6	19.0	19.1	19.3	19.2	18.9	19.0	19.1	19.0	19.0	196.2
ModelC (log 2D) MultiHot KLD	24.3	18.9	18.9	31.3	21.4	19.0	26.8	19.5	19.2	19.2	218.4
ModelC (log 2D) OneHot BCE	23.0	23.3	25.6	23.9	19.4	20.0	19.5	20.0	19.3	20.2	214.2
ModelC (log 2D) OneHot KLD	24.5	26.4	28.2	33.8	22.8	27.8	30.0	20.7	38.0	26.7	278.7

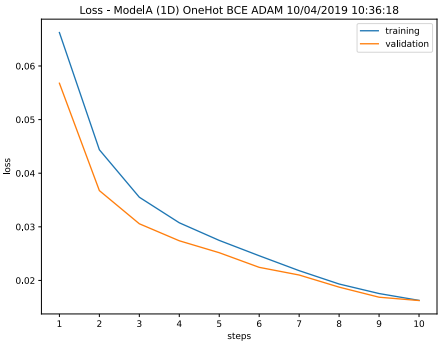
The duration of each epoch as well as total time taken for 10 epochs in minutes. For different models using ADAM optimisation.

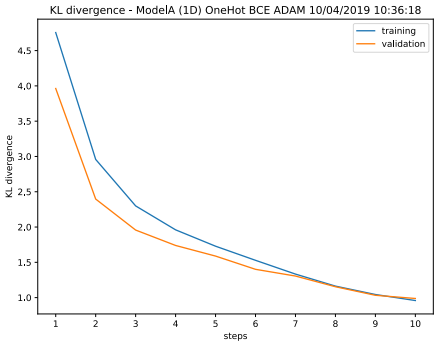
C.3 Model A training metrics

Referred to in section 4.1.1

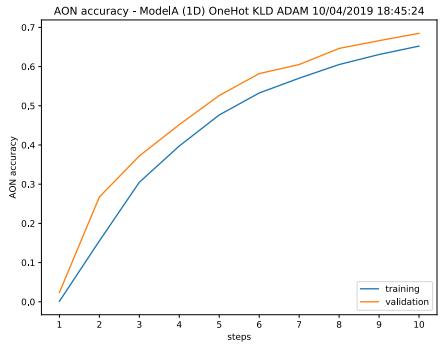


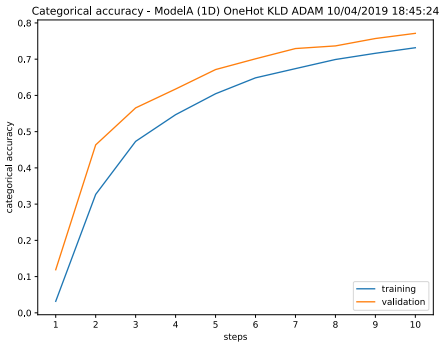


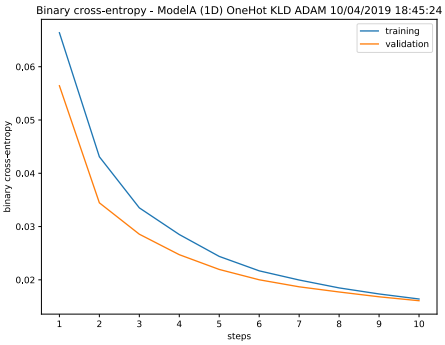


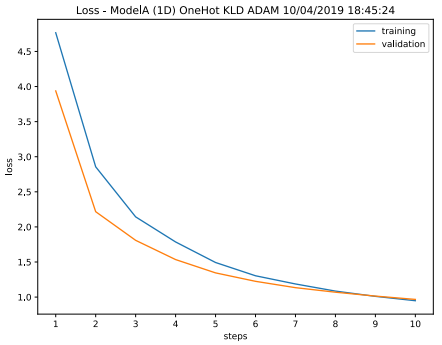


One-hot encoding, BCE loss

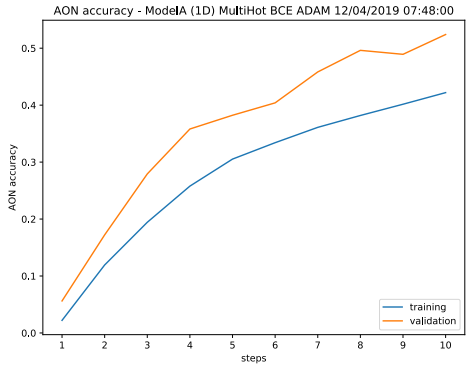


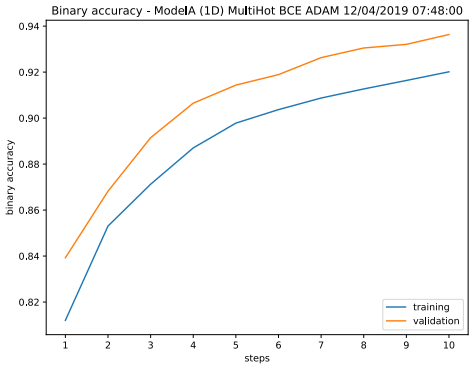


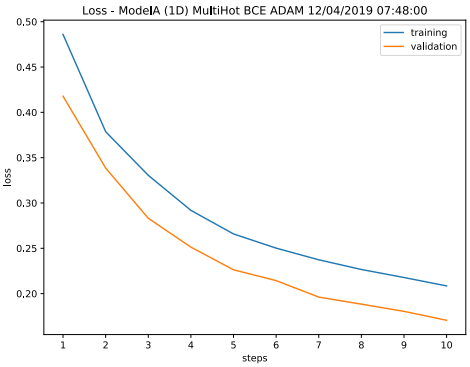


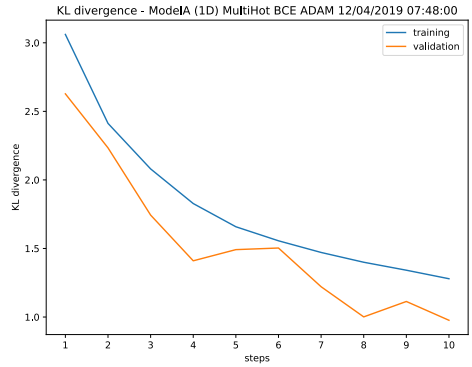


One-hot encoding, KLD loss

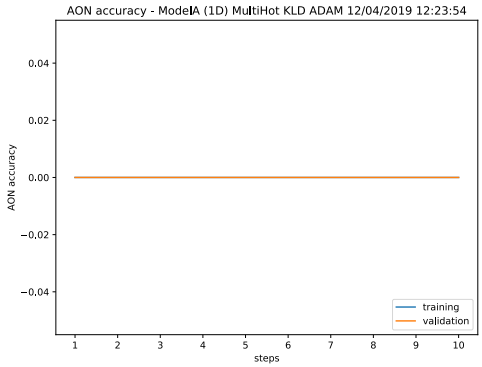


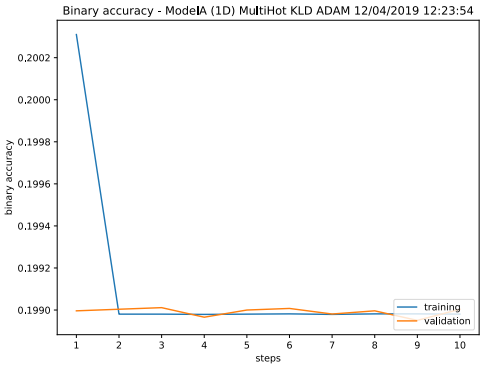


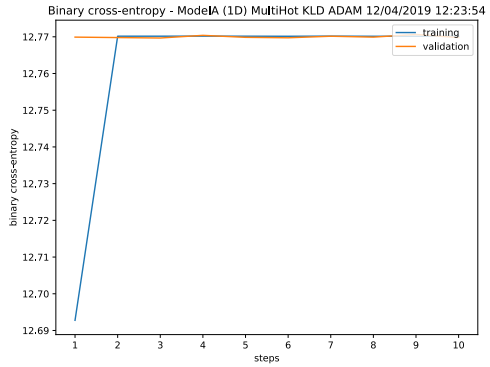


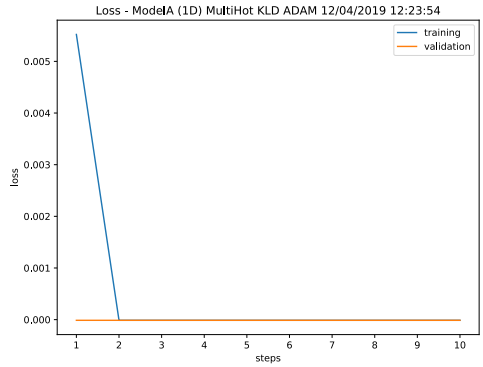


Multi-hot encoding, BCE loss





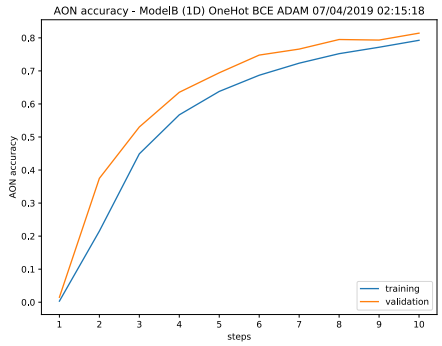


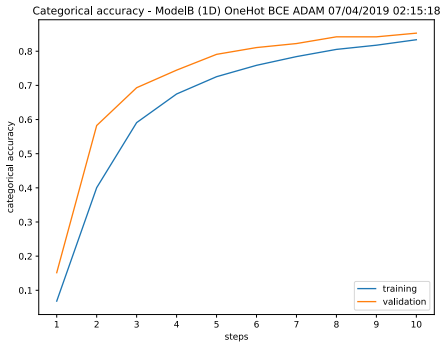


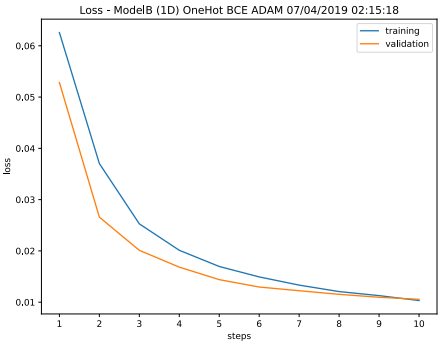
Multi-hot encoding, KLD loss

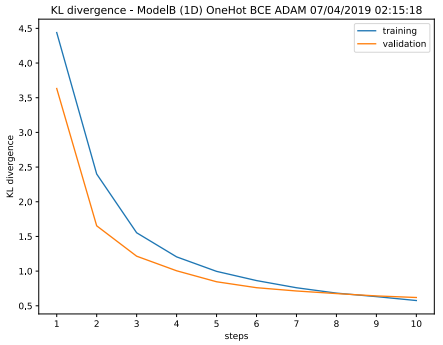
C.4 Model B training metrics

Referred to in section 4.1.2

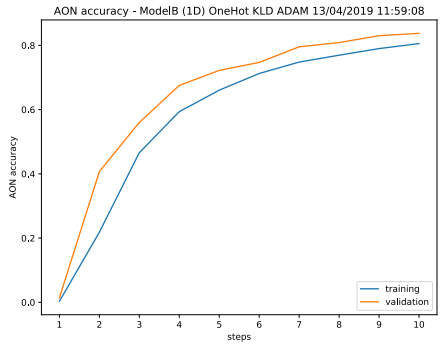


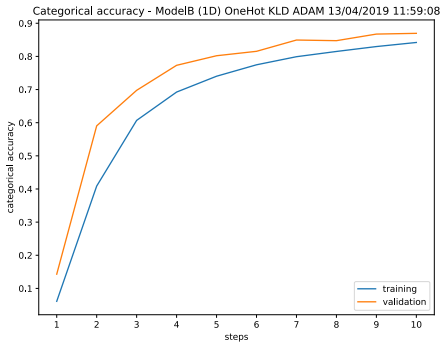


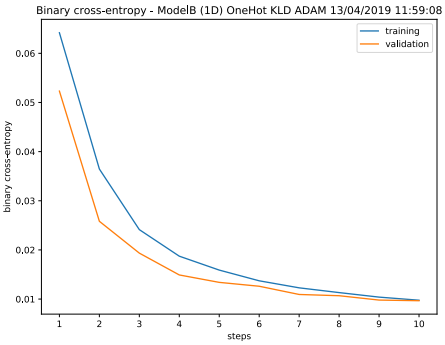


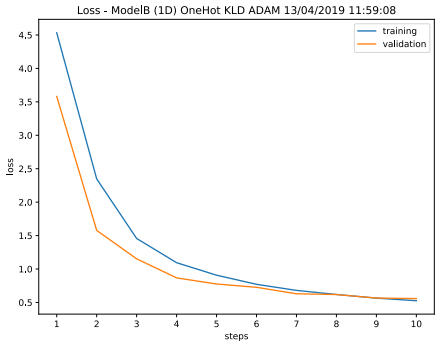


One-hot encoding, BCE loss

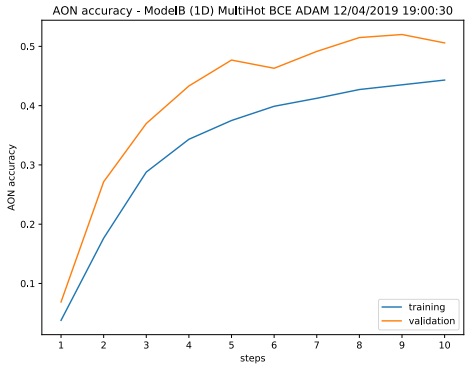


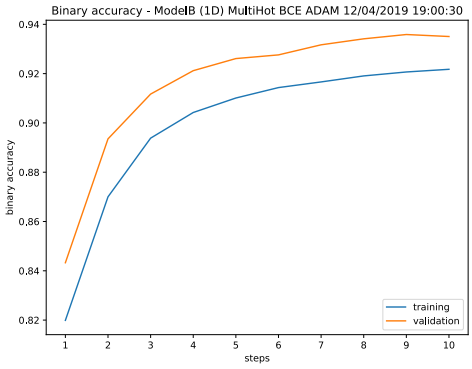


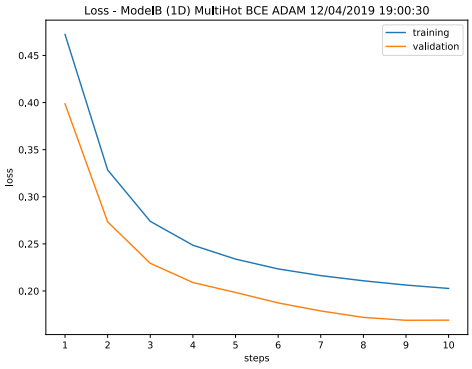


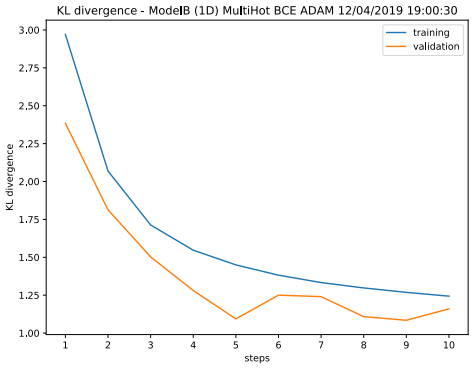


One-hot encoding, KLD loss

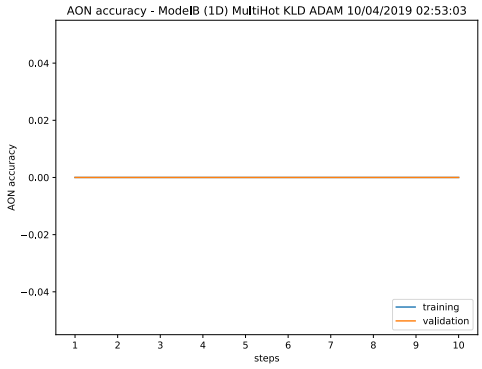


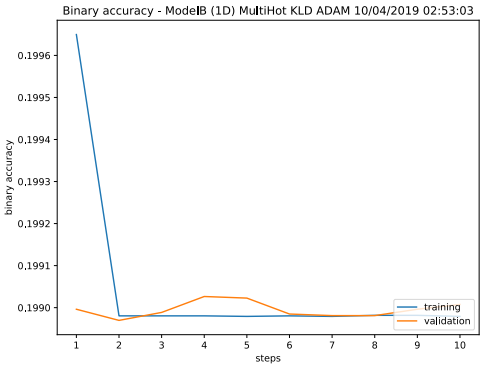


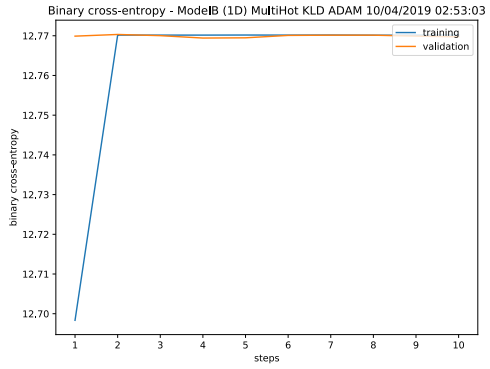


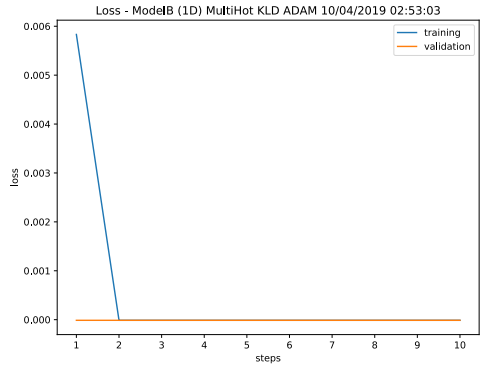


Multi-hot encoding, BCE loss





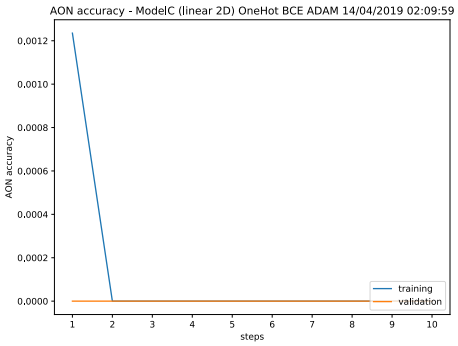


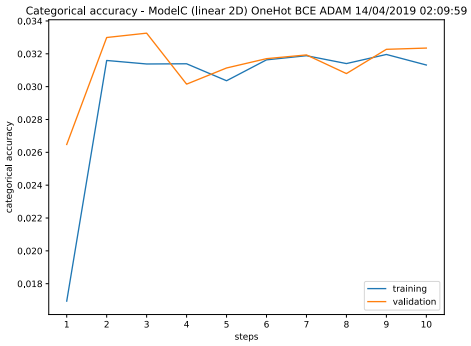


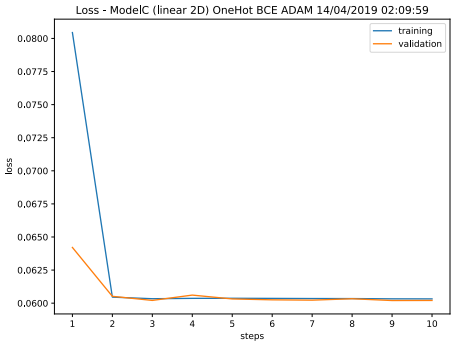
Multi-hot encoding, KLD loss

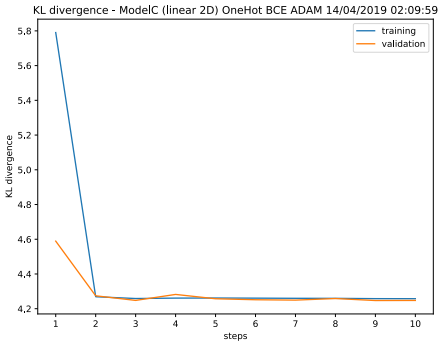
C.5 Model C training metrics

Referred to in section 4.1.3

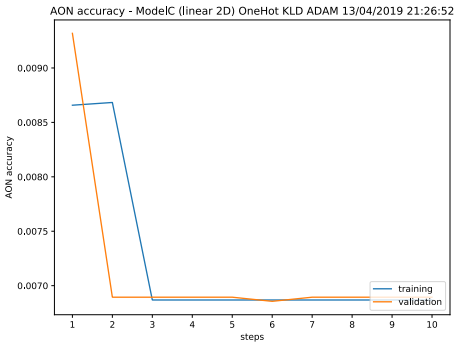


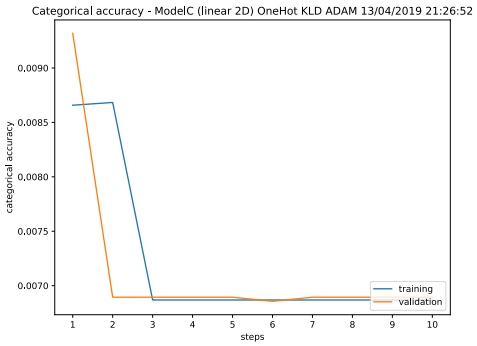


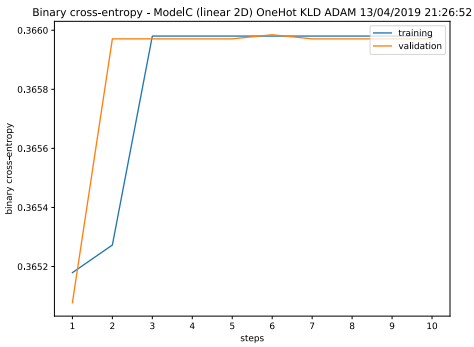


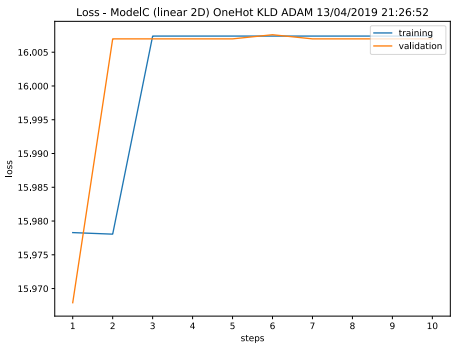


Linear magnitude, one-hot encoding, BCE loss

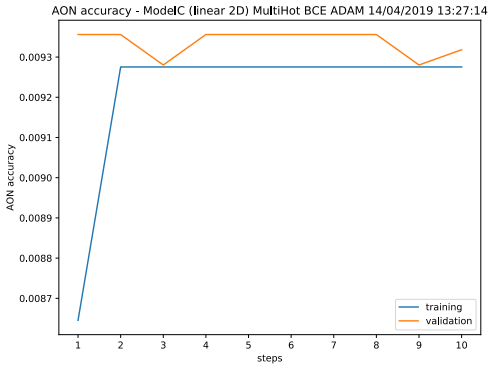


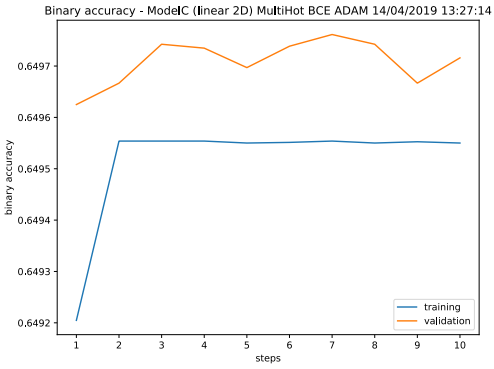


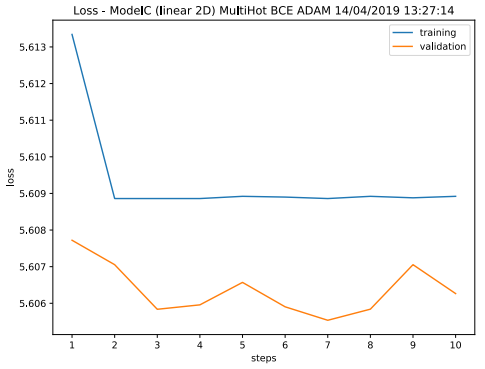


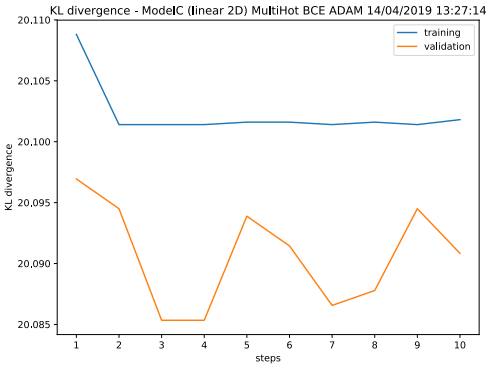


Linear magnitude, one-hot encoding, KLD loss

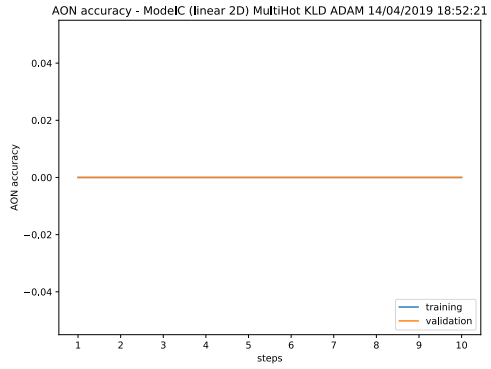


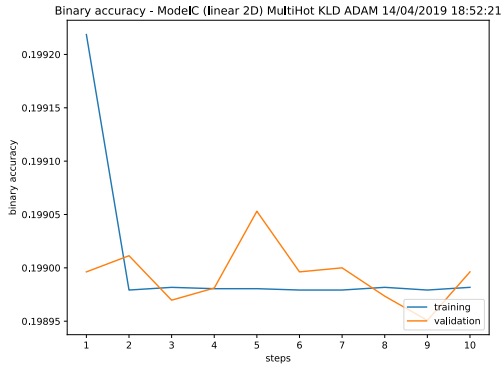


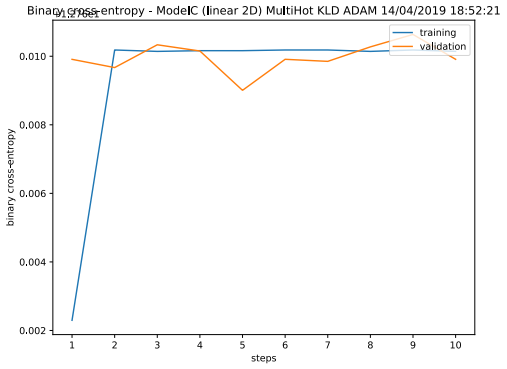


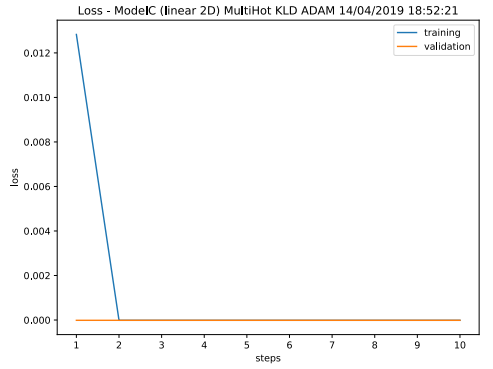


Linear magnitude, multi-hot encoding, BCE loss

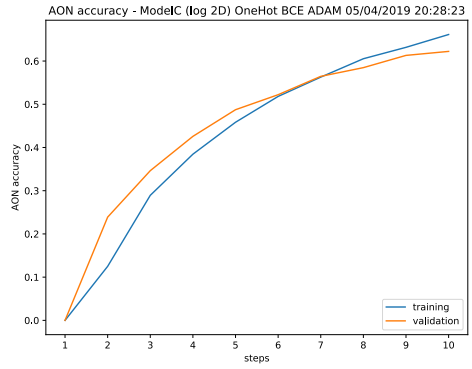


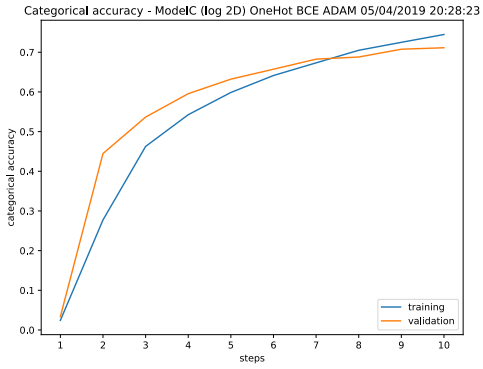


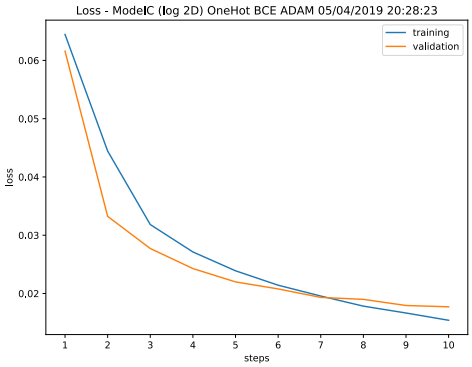


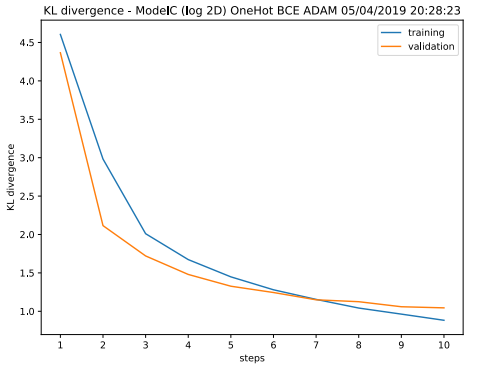


Linear magnitude, multi-hot encoding, KLD loss

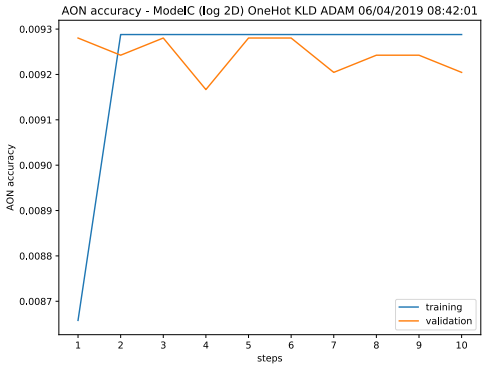


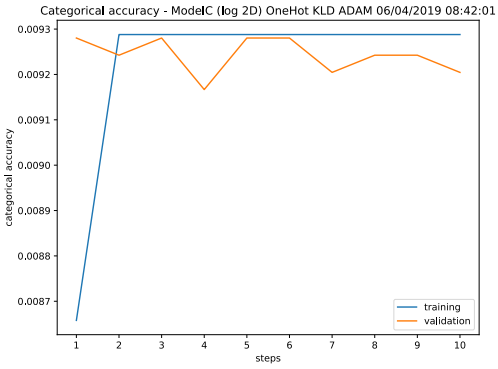


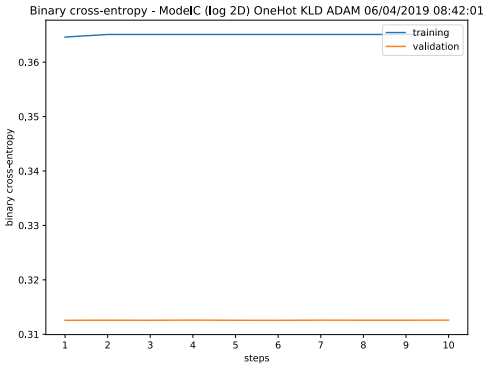


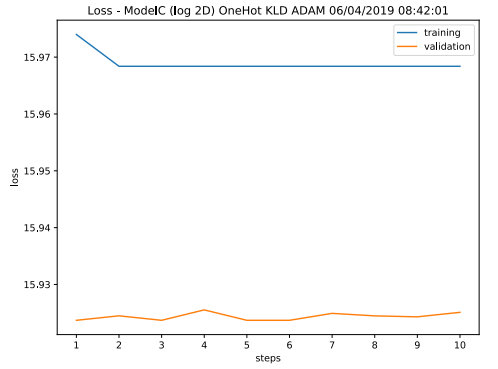


Log magnitude, one-hot encoding, BCE loss

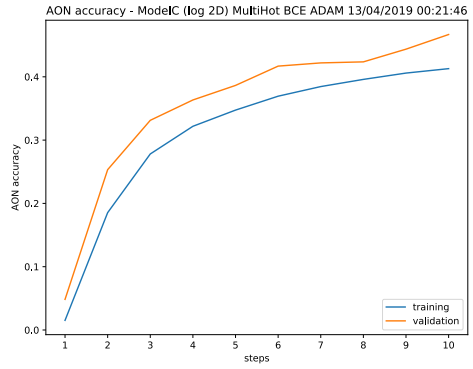


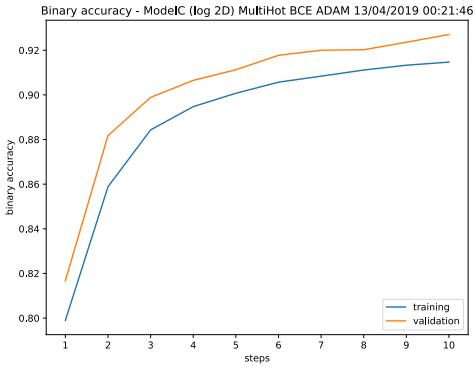


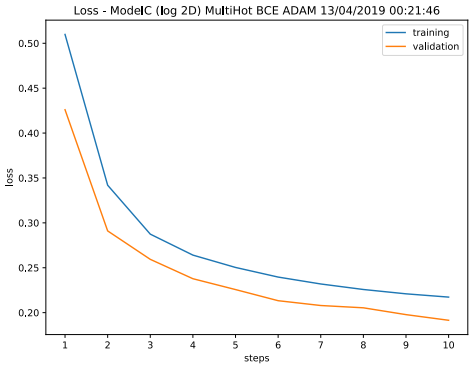


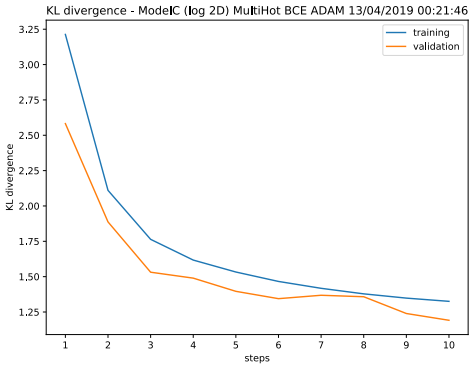


Log magnitude, one-hot encoding, KLD loss

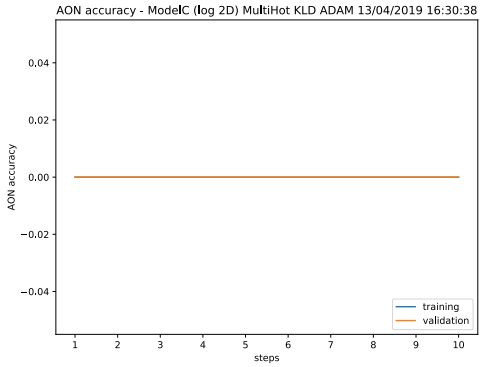


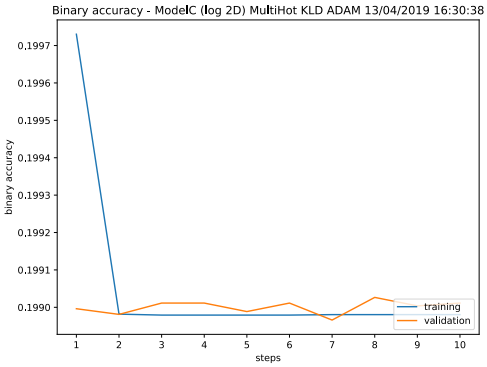


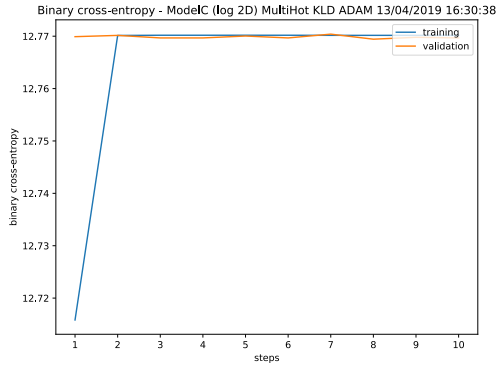


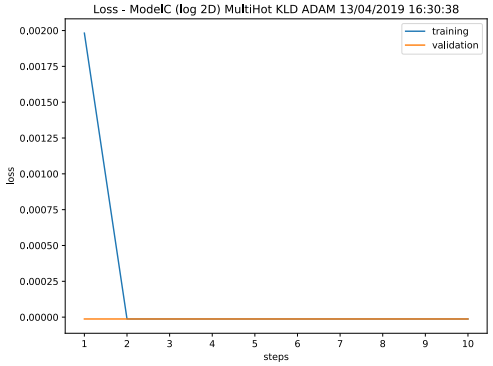


Log magnitude, multi-hot encoding, BCE loss







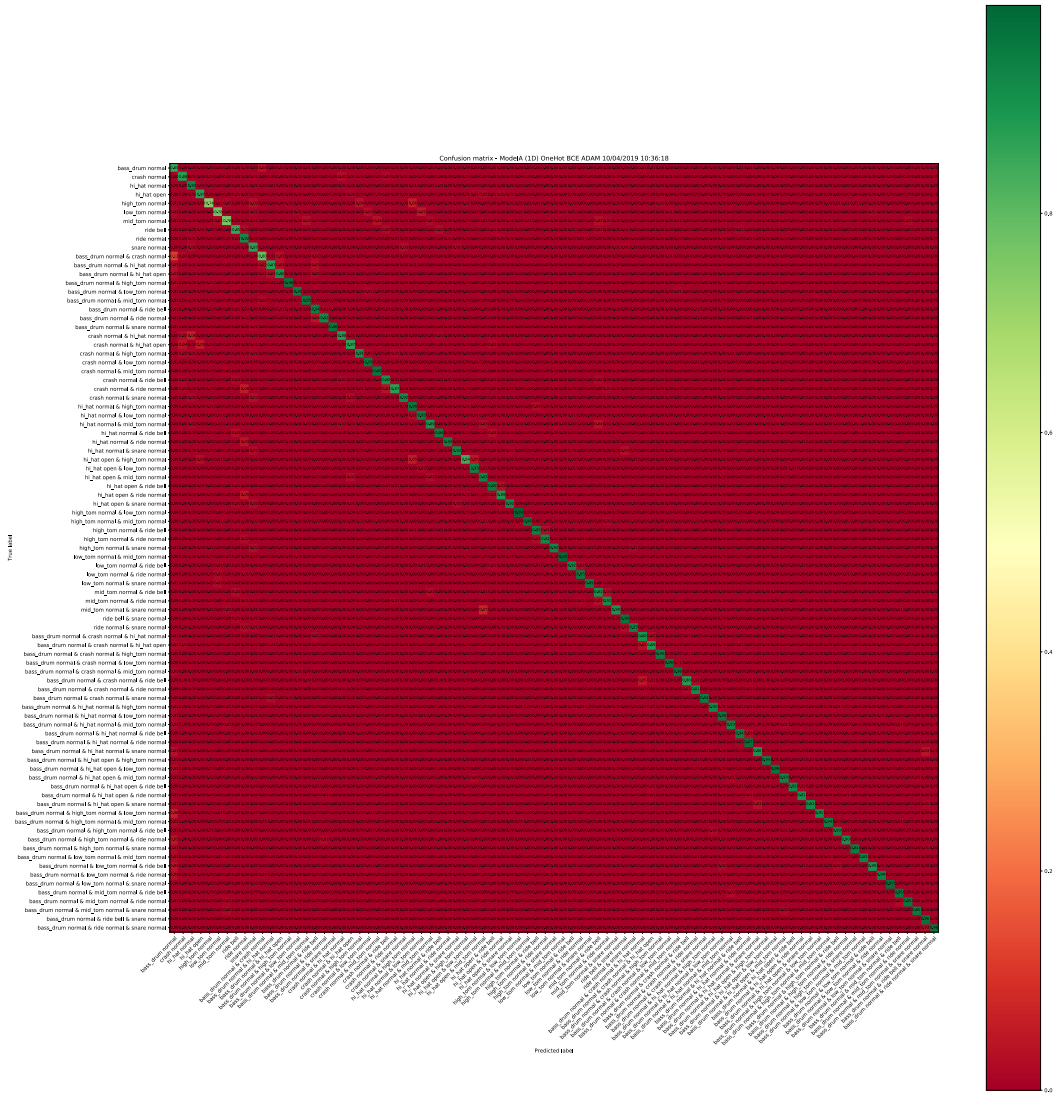


Log magnitude, multi-hot encoding, KLD loss

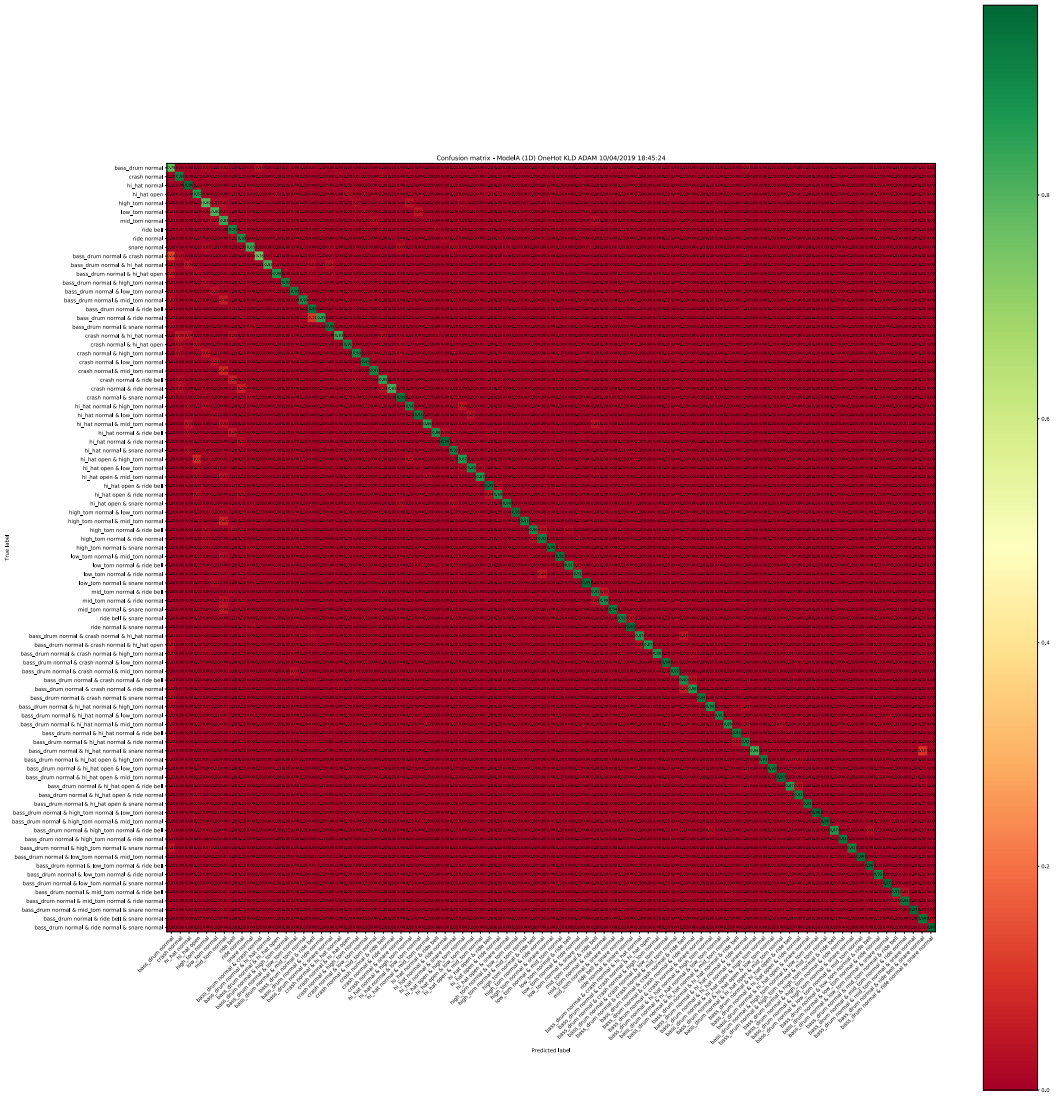
D Confusion matrices

D.1 Model A

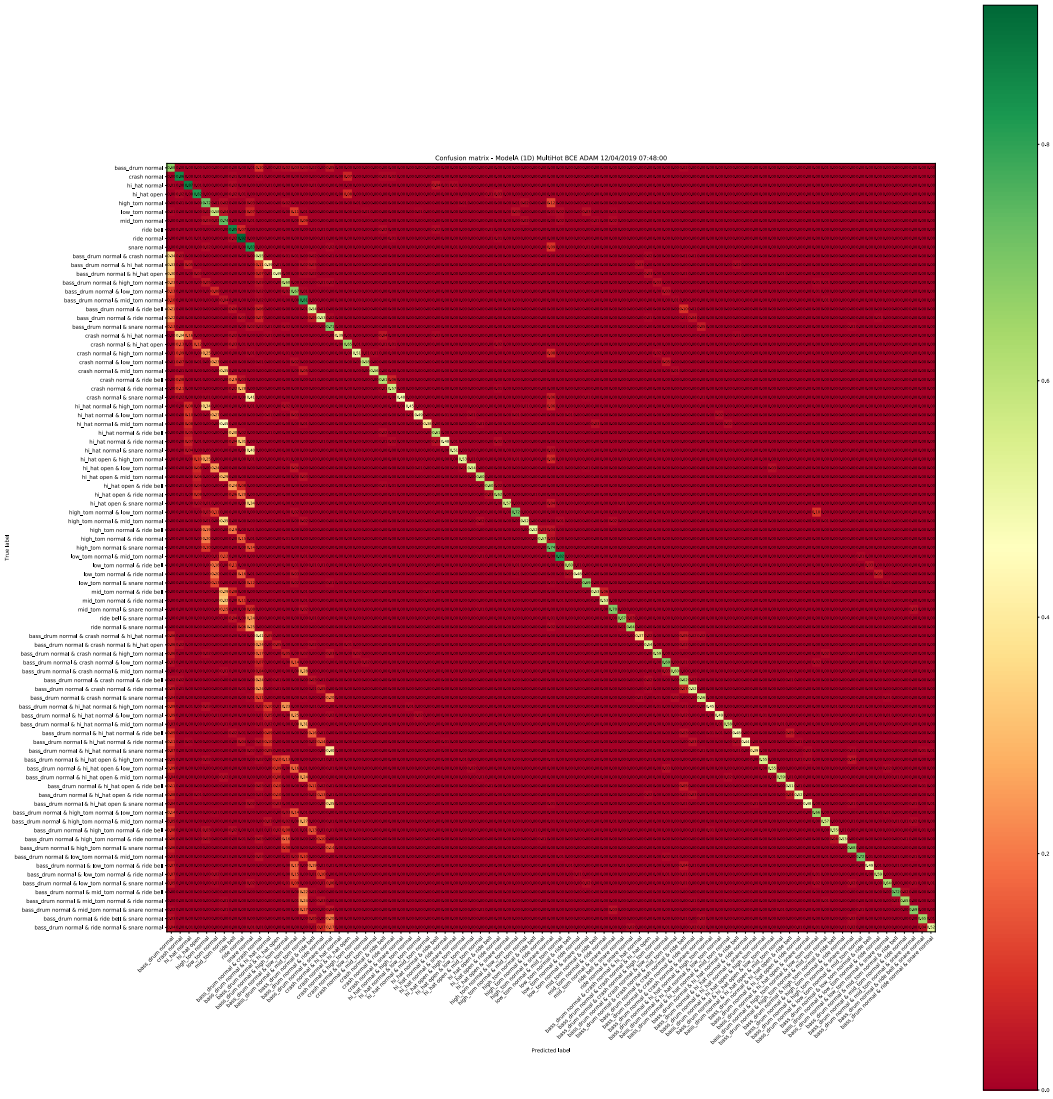
Referred to in section 4.2.2



Model A with one-hot encoding and BCE loss. This model got: 68.62% Acc_{AON} and left 25.69% of test data unclassified.



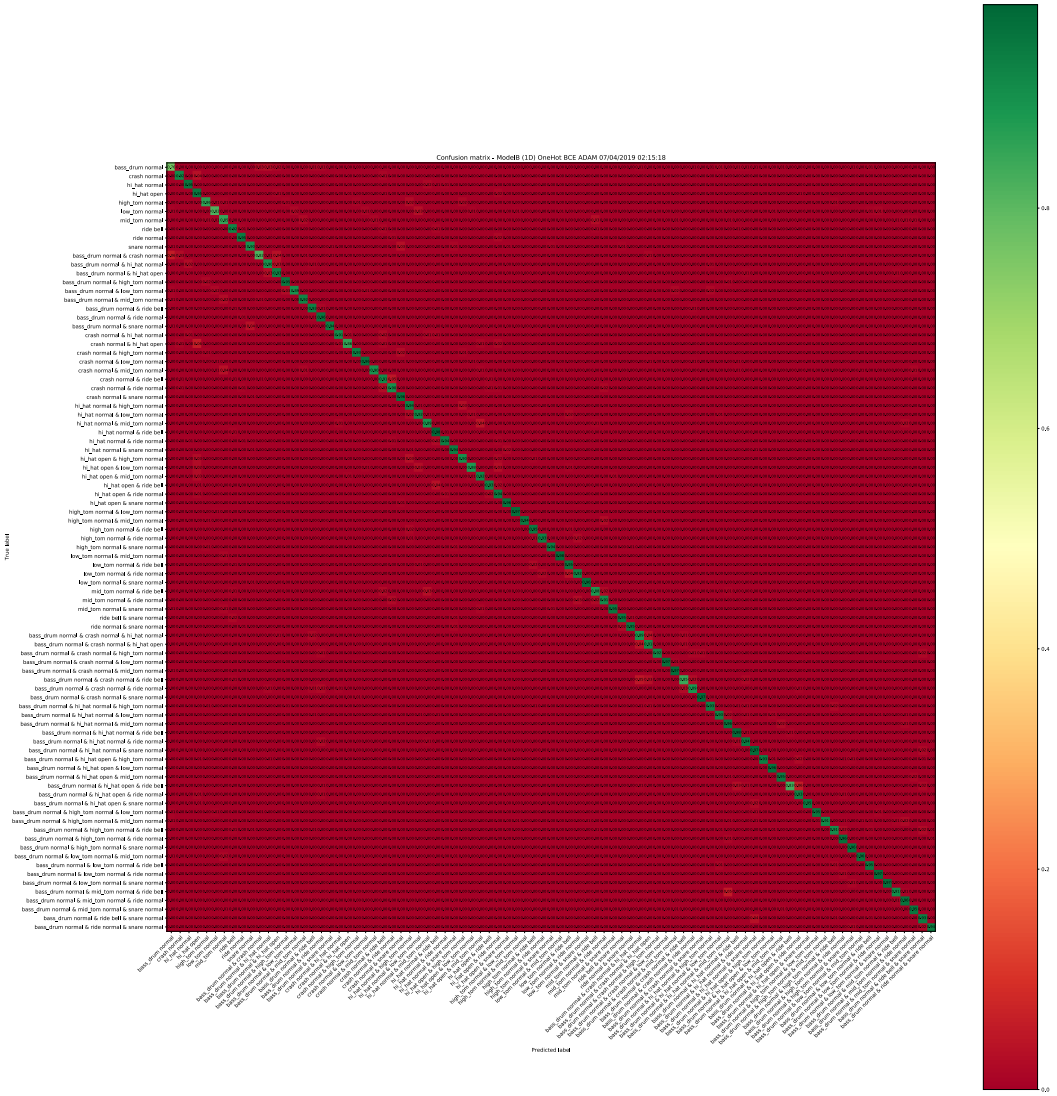
Model A with one-hot encoding and KLD loss. This model got: 69.30% Acc_{AON} and left 24.11% of test data unclassified.



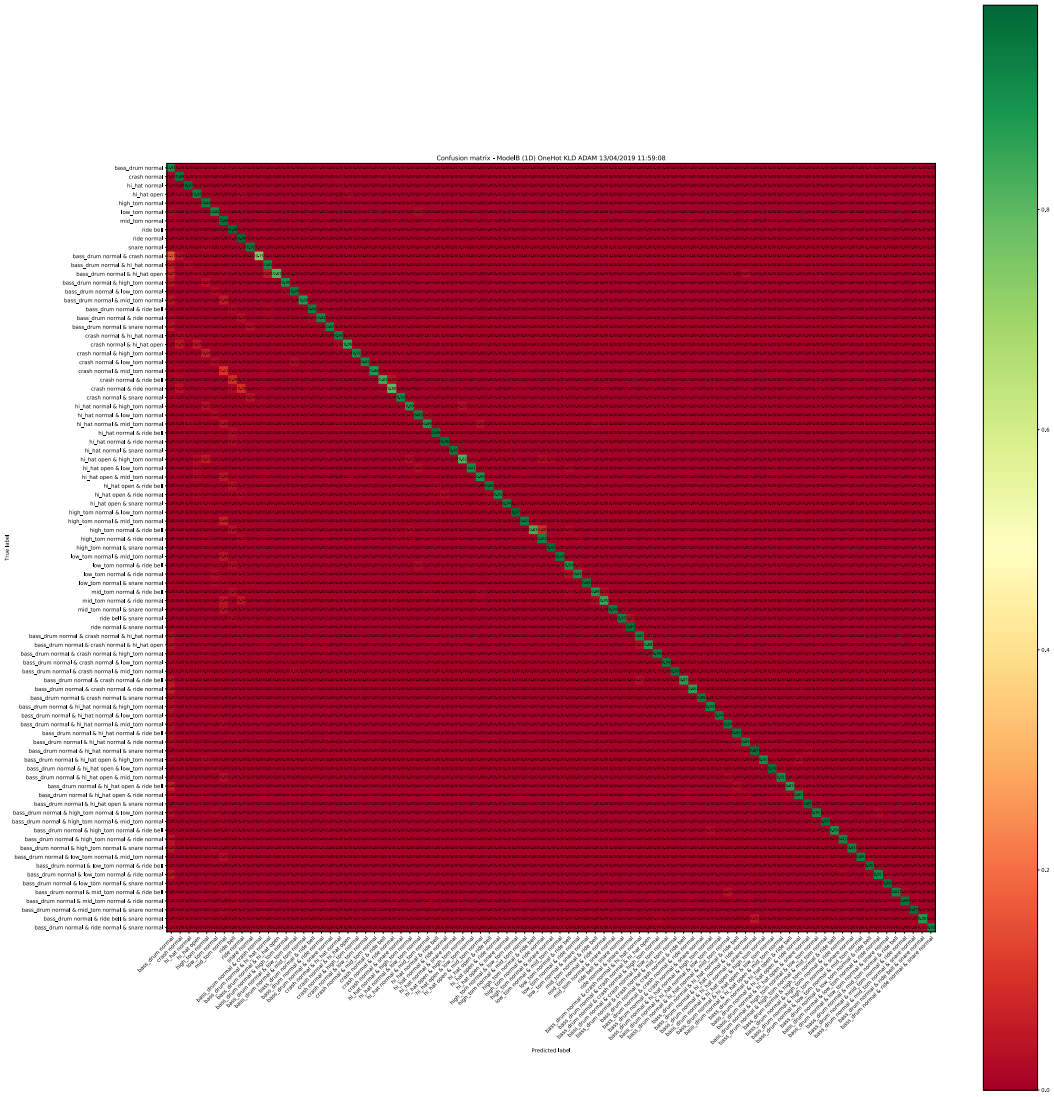
Model A with multi-hot encoding and BCE loss. This model got: 52.53% Acc_{AON} and left 19.77% of test data unclassified.

D.2 Model B

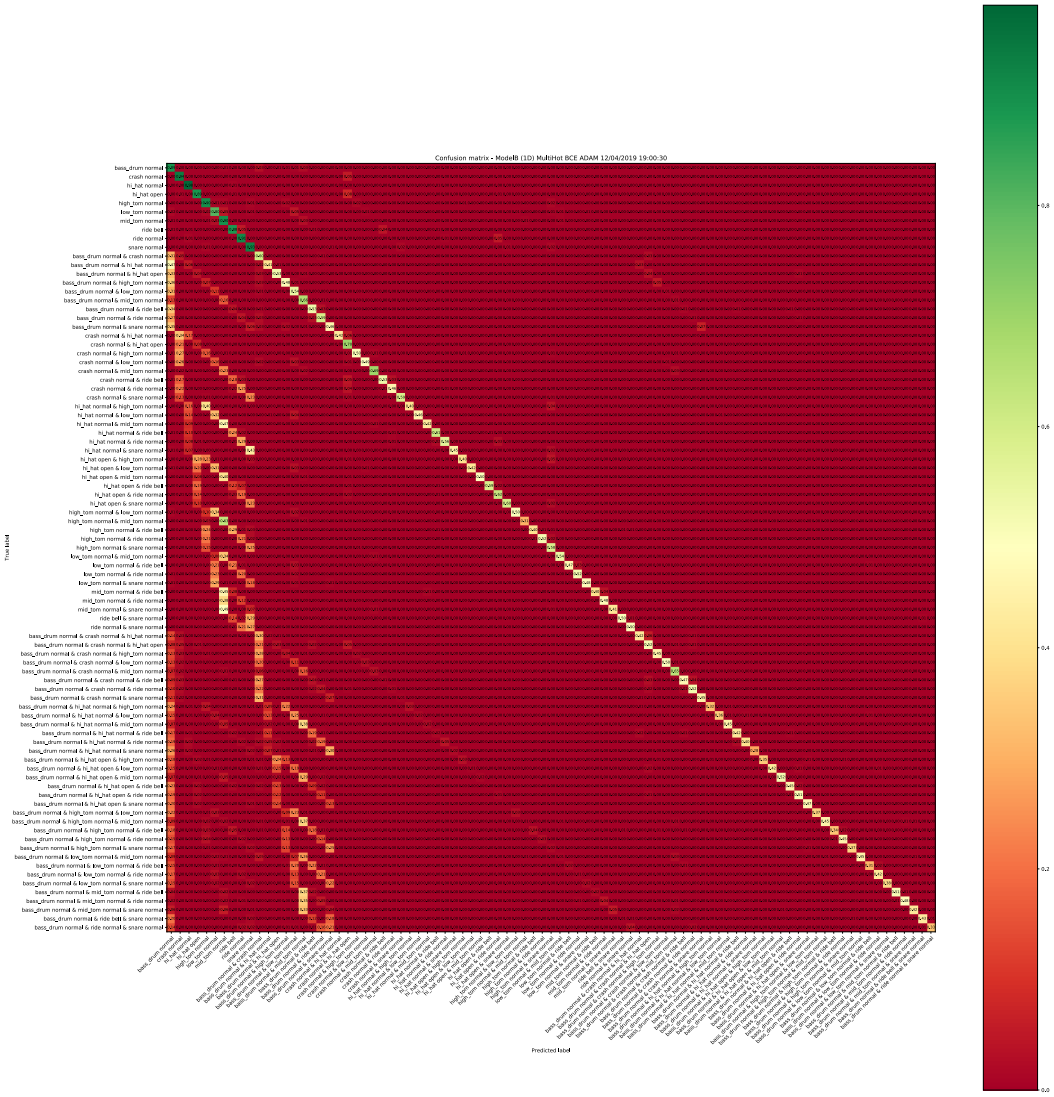
Referred to in section 4.2.3



Model B with one-hot encoding and BCE loss. This model got: 81.53% Acc_{AON} and left 12.74% of test data unclassified.



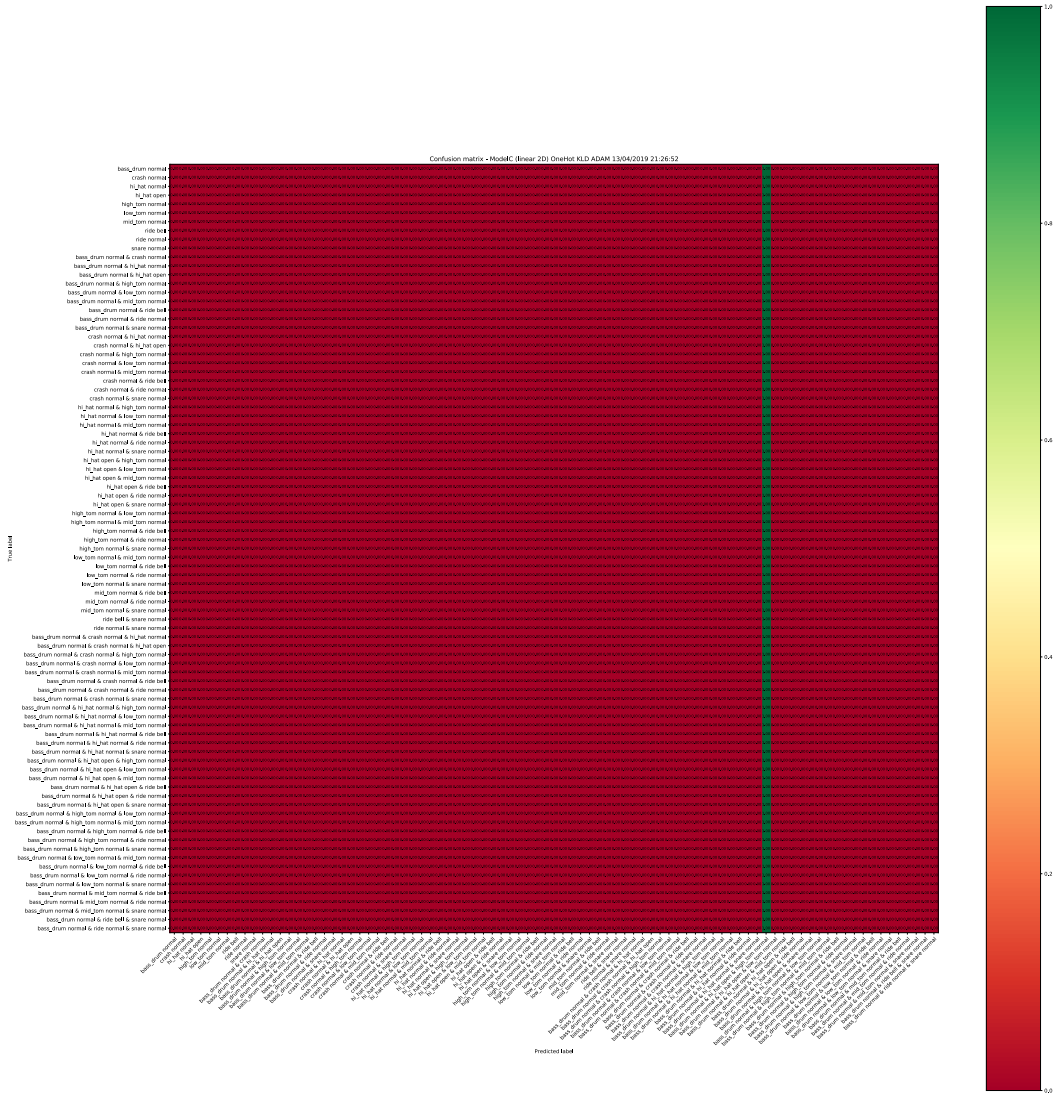
Model B with one-hot encoding and KLD loss. This model got: 83.80% Acc_{AON} and left 12.38% of test data unclassified.



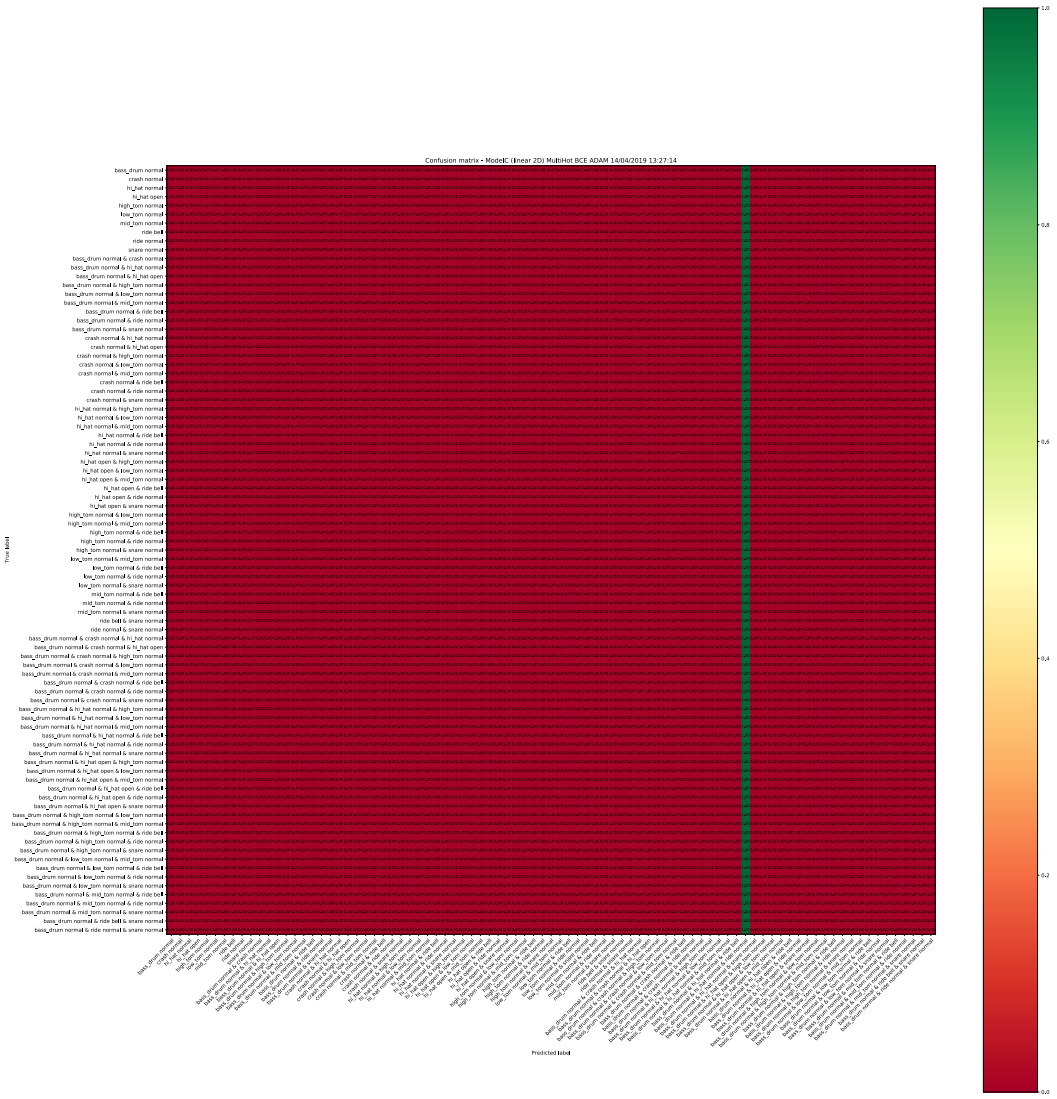
Model B with multi-hot encoding and BCE loss. This model got: 50.67% Acc_{AON} and left 21.89% of test data unclassified.

D.3 Model C

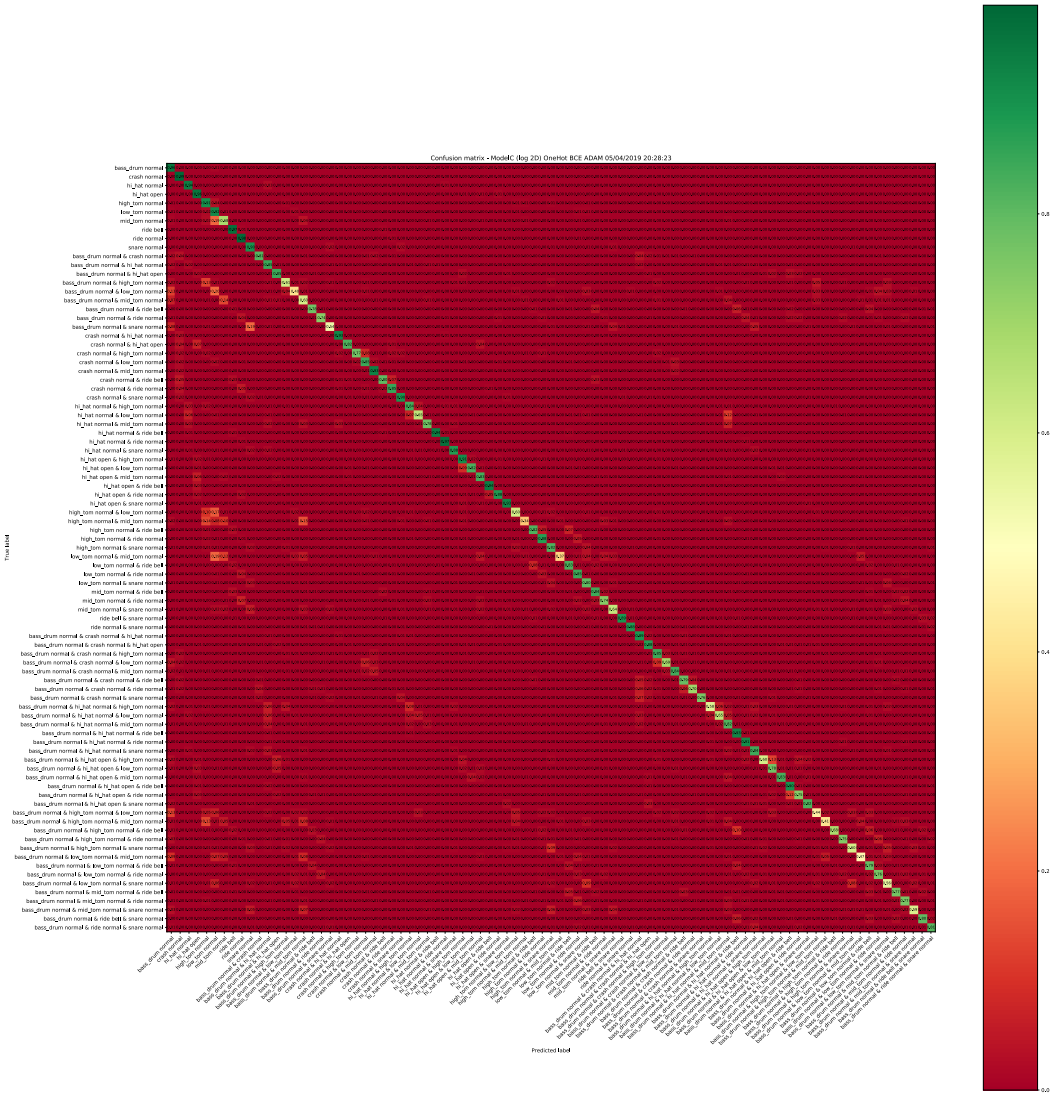
Referred to in section 4.2.4



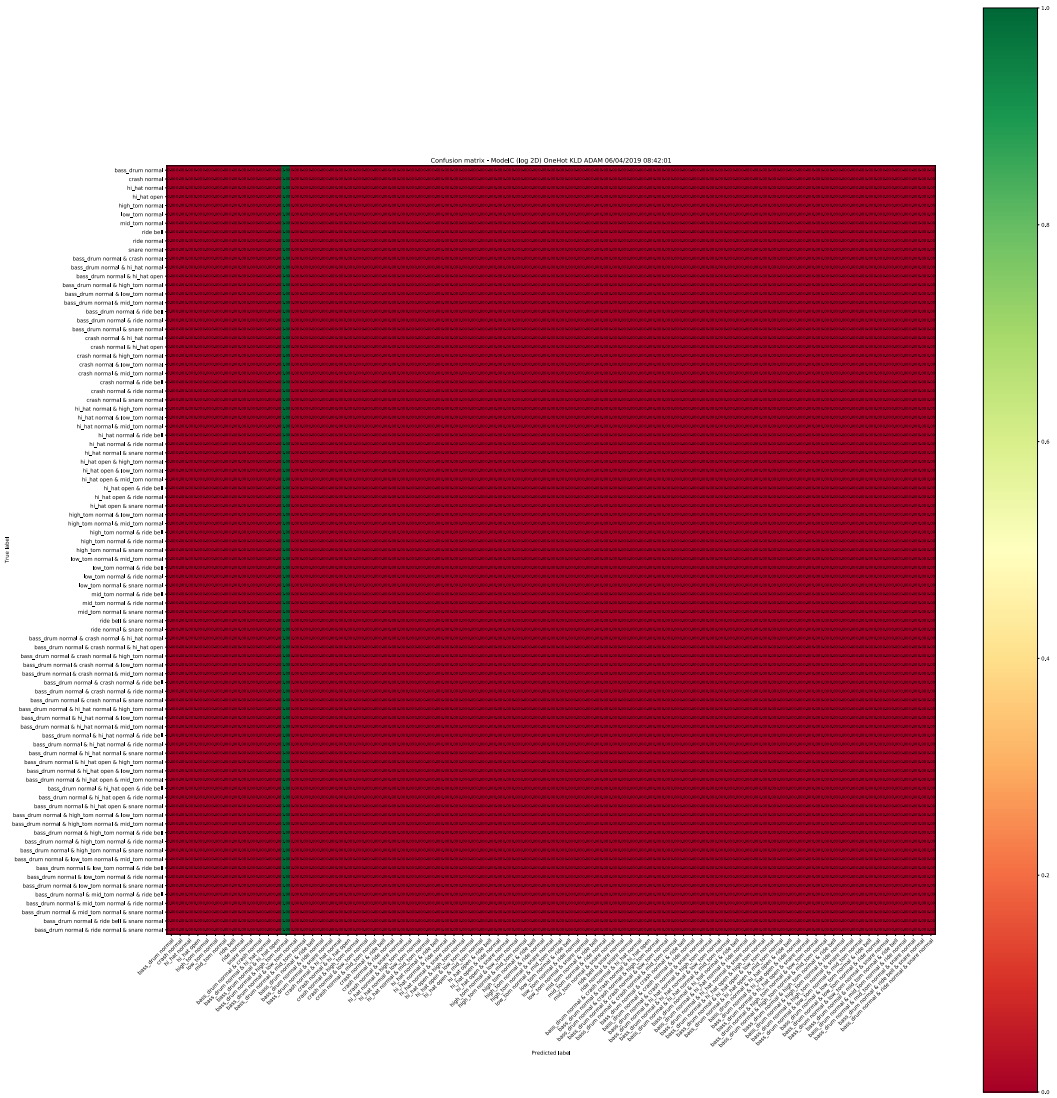
Model C using linear spectrogram input with one-hot encoding and KLD loss. This model got: 0.69% Acc_{AON} and left 0.00% of test data unclassified.



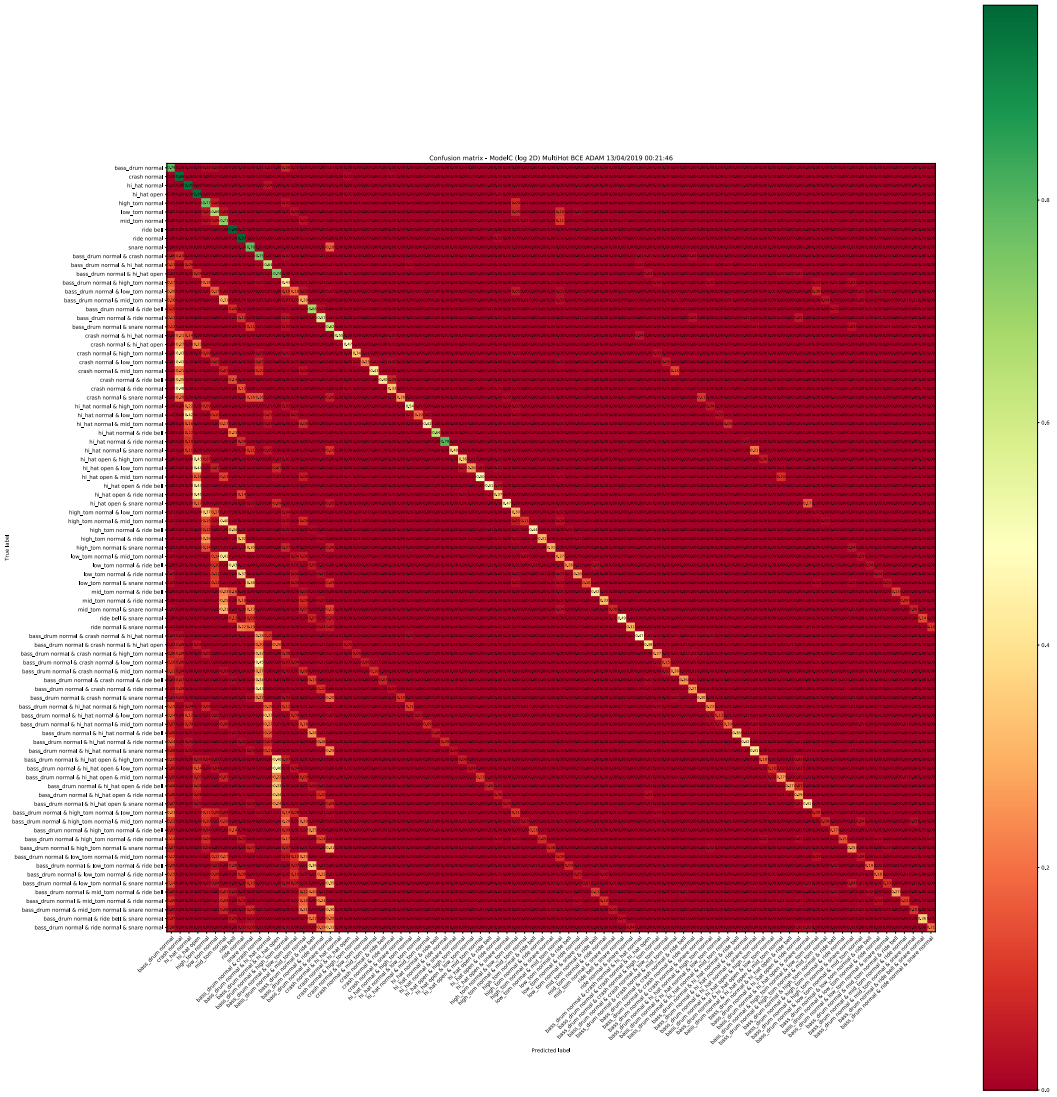
Model C using linear spectrogram input with multi-hot encoding and BCE loss. This model got: 0.93% Acc_{AON} and left 0.00% of test data unclassified.



Model C using log spectrogram input with one-hot encoding and BCE loss. This model got: 62.64% Acc_{AON} and left 29.06% of test data unclassified.



Model C using log spectrogram input with one-hot encoding and KLD loss. This model got: 00.64% Acc_{AON} and left 0.00% of test data unclassified.



Model C using log spectrogram input with multi-hot encoding and BCE loss. This model got: 46.91% Acc_{AON} and left 10.67% of test data unclassified.