Convolutioanl Neural Network models for audio infilling and prediction tasks

Student ID: 200576260

Contents

1 Overview

This report demonstrates the training of two different types of convolutional neural network models on an audio dataset that had been transformed into images of mel-spectrograms as seen in [1]. The data was masked in the middle, as inspired by papers such as [2] for audio inpainting/infilling, and masked at the end to see if these networks would be suitable for audio/music prediction tasks. The results were not satisfactory, and it was concluded that the architecture of the models used were either not complex enough to pick up the various features of the audio data, or the model and loss functions used were the wrong type for the task.

2 Data

The models were trained separately for separate tasks using the Saarland Music Dataset (SMD)[3], which consists of over 9.2 hours of piano audio and midi data. The dataset was chosen specifically as the recordings were all recorded on the same instrument, with the same recording settings. For these two tasks, only the audio data was used from the dataset. The data was used for the x-input and the y-prediction, where the x-input data was masked differently for two separate tasks. During pre-processing, the audio was segmented into 10s clips, at a sample rate of 22050Hz. The audio was then normalised, converted to mono, and each clip was transformed into a mel-spectrogram of 64 mel bins, with an fft size of 1024 over a hop length of 512. The mel-spectrogram was also normalised to help the model converge more quickly. The end dimensions of the each mel-spectrogram tensor were [1, 64, 431]. The total size of the dataset was 3325 samples.
[1]

[1]https://github.com/schultzm/SliceAudio Link to script used for slicing audio data.

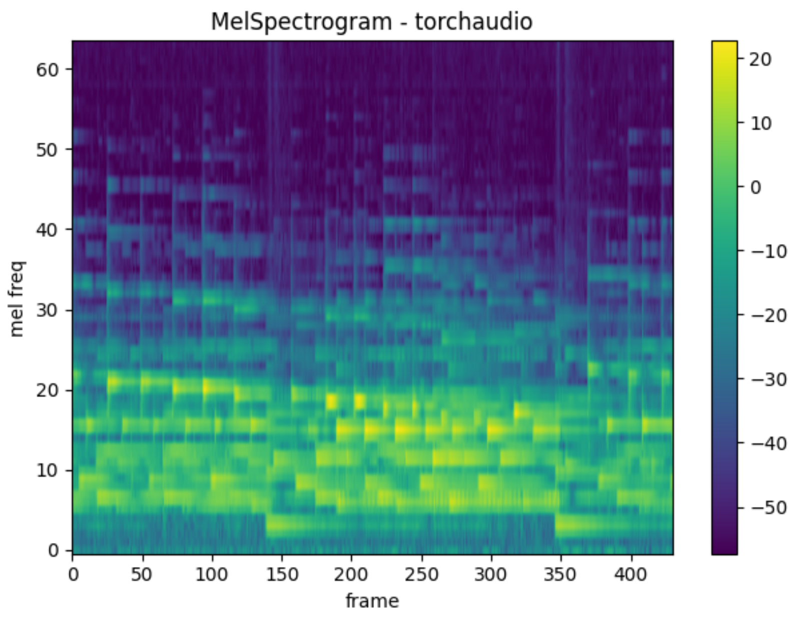


Figure 1: Example audio data as mel-spectrogram

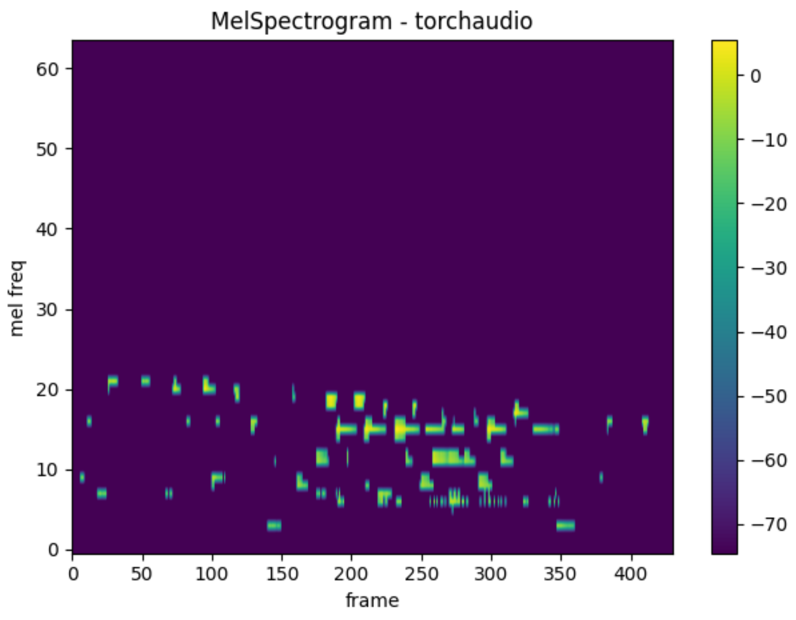


Figure 2: Example audio data as normalised mel-spectrogram

For the first task, the mel-spectrograms were masked by 10 time bins in the middle from time bin 200 to 210. This equated to approximately 0.23s seconds of audio. The masked mel-spectrograms were used as a the x-input of the dataset, and trained to predict the y-output of the same mel-spectrogram, but unmasked.

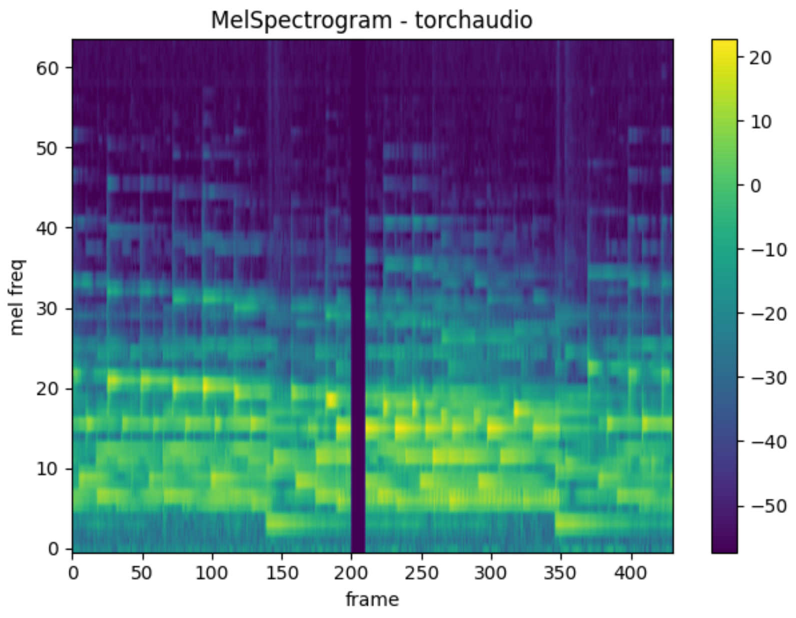


Figure 3: Example audio data as a mel-spectrogram, with masked middle of 10 time bins

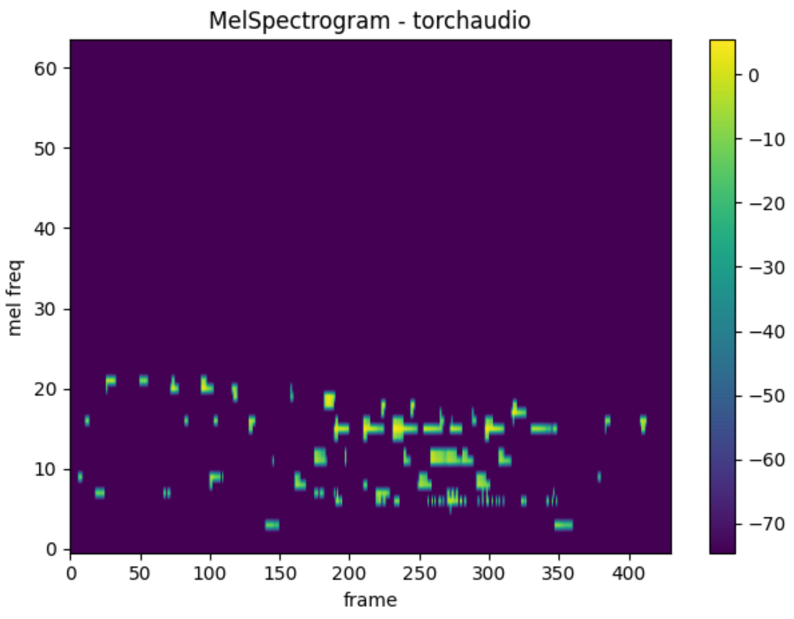


Figure 4: Example audio data as normalised mel-spectrogram, with masked middle of 10 time bins

For the second task, the mel-spectrograms were masked by 20 time bins at the end of the mel-spectrograms. These masked mel-spectrograms were used as the x-input of the dataset, and were also trained to predict the y-output of the unmasked mel-spectrogram of the data.

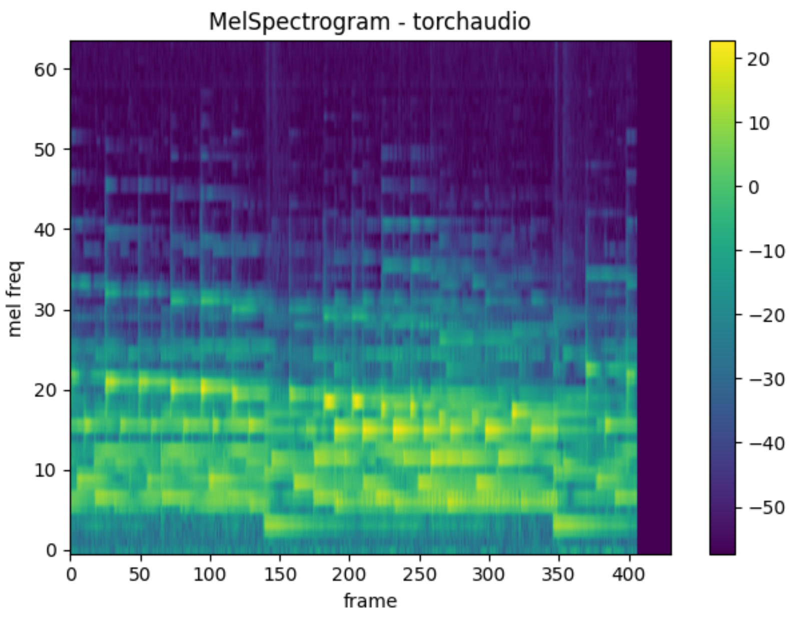


Figure 5: Example audio data mel-spectrogram, with masked end of 20 time bins

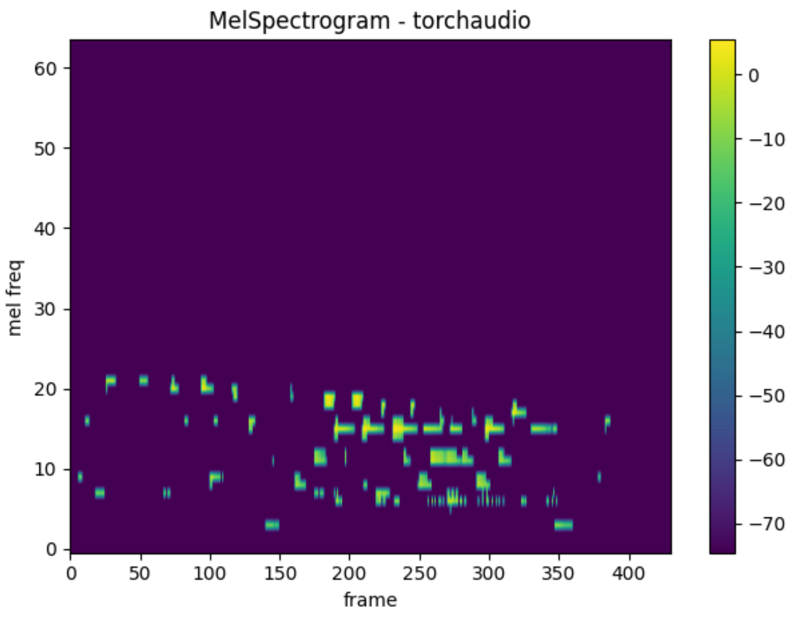
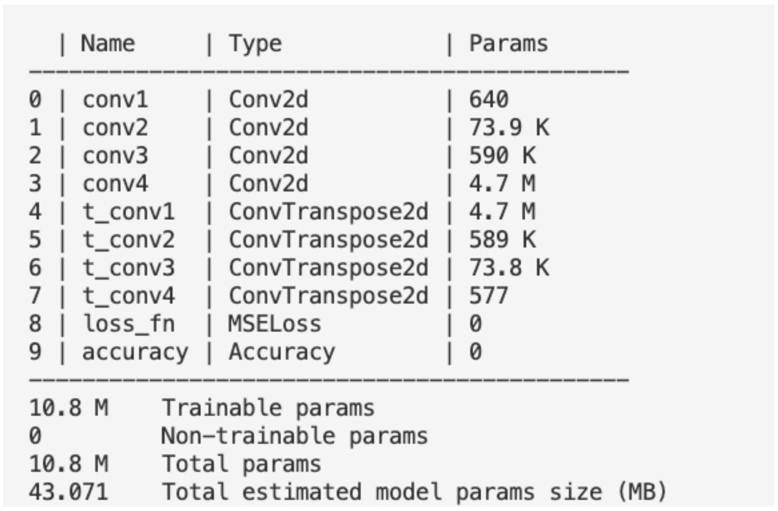


Figure 6: Example audio data as normalised mel-spectrogram, with masked end of 20 time bins

3 Model Designs

3.1 Task 1: Audio Infilling using an Autoencoder with Convolutional layers

The first task consisted of getting a model to learn the masked middle section of the mel-spectrogram data. The idea for masking the middle of the training data came from audio infilling tasks, where perhaps some audio data had been lost in the process of editing, and needed to be filled in. For this, an autoencoder model with convoutional layers was chosen. The model consisted of 10.8 million trainable parameters, and the estimated model size was 43.071MB. The parameter size of the model was quite large for an autoencoder, as no MaxPooling layers were used. The architecture consisted of three convolutional layers, followed by three transposed convolutional layers. The loss function used was Mean Squared Error (MSE).



```
  | Name     | Type            | Params
---------------------------------------------
0 | conv1    | Conv2d          | 640
1 | conv2    | Conv2d          | 73.9 K
2 | conv3    | Conv2d          | 590 K
3 | conv4    | Conv2d          | 4.7 M
4 | t_conv1  | ConvTranspose2d | 4.7 M
5 | t_conv2  | ConvTranspose2d | 589 K
6 | t_conv3  | ConvTranspose2d | 73.8 K
7 | t_conv4  | ConvTranspose2d | 577
8 | loss_fn  | MSELoss         | 0
9 | accuracy | Accuracy        | 0
---------------------------------------------
10.8 M    Trainable params
0         Non-trainable params
10.8 M    Total params
43.071    Total estimated model params size (MB)
```

Figure 7: Architecture of the Convolutional Autoencoder model

3.2 Task 2: Audio Prediction using a Variational Autoencoder with Convolutional layers

For the second task, a Variational Autoencoder (VAE) was used to train 3325 end masked samples of audio data. A more complex model was used for the second task, as it seemed appropriate to try and get new predictions from the mappings made by the model. The model was larger than the Autoencoder model, with 36.6million trainable parameters and a model size of approximately 146.27MB. The architecture of the model consisted of three convolutional layers followed by a bottleneck of fully connected layers of z-dimensional space (probability distribution of the compressed data), which the convoluted data was fed into. The data was decompressed through three transposed convolutional layers. Initially, MSE loss function was used, but it was eventually changed to KL-divergence loss. A lot of difficulty was experienced in finding the right combinations of layers, which would give the desired dimensions for the output. Various configurations of layers were experimented with, including MaxPooling and Dropout in order to reduce the amount of parameters in the hidden state - however the correct values for kernal size, padding and stride could not be found using these combinations, resulting in the wrong output dimensions which meant that the model would not train. Eventually the model trained with a large hidden state size.

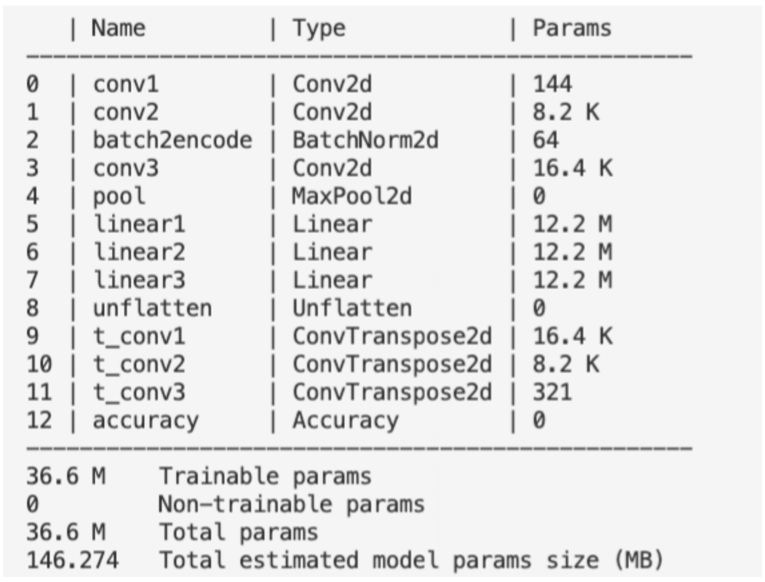

```
   | Name          | Type             | Params
------------------------------------------------
0  | conv1         | Conv2d           | 144
1  | conv2         | Conv2d           | 8.2 K
2  | batch2encode  | BatchNorm2d      | 64
3  | conv3         | Conv2d           | 16.4 K
4  | pool          | MaxPool2d        | 0
5  | linear1       | Linear           | 12.2 M
6  | linear2       | Linear           | 12.2 M
7  | linear3       | Linear           | 12.2 M
8  | unflatten     | Unflatten        | 0
9  | t_conv1       | ConvTranspose2d  | 16.4 K
10 | t_conv2       | ConvTranspose2d  | 8.2 K
11 | t_conv3       | ConvTranspose2d  | 321
12 | accuracy      | Accuracy         | 0
------------------------------------------------
36.6 M    Trainable params
0         Non-trainable params
36.6 M    Total params
146.274   Total estimated model params size (MB)
```

Figure 8: Architecture of the Convolutional Variational Autoencoder model

4 Implementation

4.1 Task 1

The AE-CNN model was trained on the masked data for tasked 1. The final model was trained for 100 epochs using MSE loss and a learning rate of 1e-4. Various iterations of the model were trained using different hyperparameters, including modifying batch size, learning rate and the type of loss function used.

4.2 Task 2

The VAE-CNN model was trained on the masked data for task 2. The model was trained for 100 epochs using KL-Divergence loss with an learning rate of 1e-4. The validation loss decreased to 0.00235 over these 100 epochs, and the outputs were tested through an inference script using the testing dataset.

5 Evaluation

An audio sample from the testing dataset was inputted into the trained autoencoder model during inference. The test data sample was masked in the middle in the same way as the masked training dataset. The output of the model was illustrated in a mel-spectrogram, which showed that the model had overfitted heavily due to the mel-frequencies all being the same colour. The model was not complex enough to be able to pick up any distinguishable features, and had heavily overfitted on a specific mel frequency.

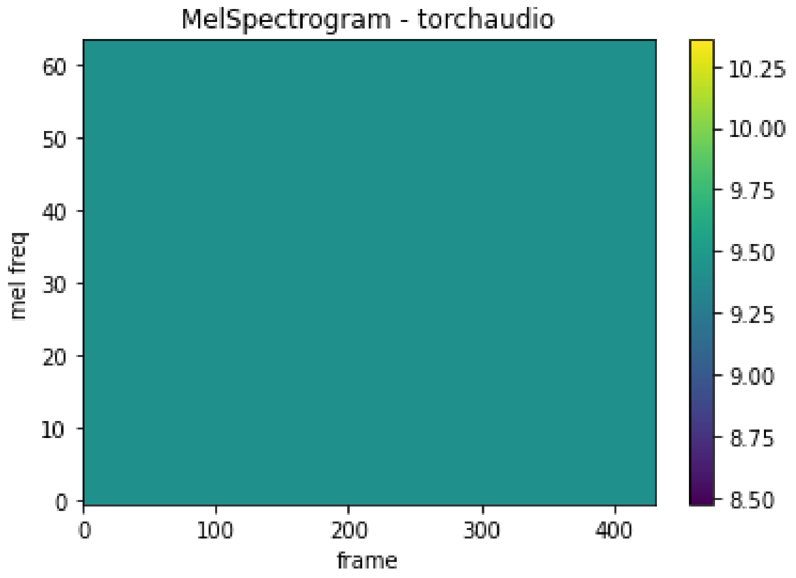


Figure 9: Test data output after forward pass through trained AE model

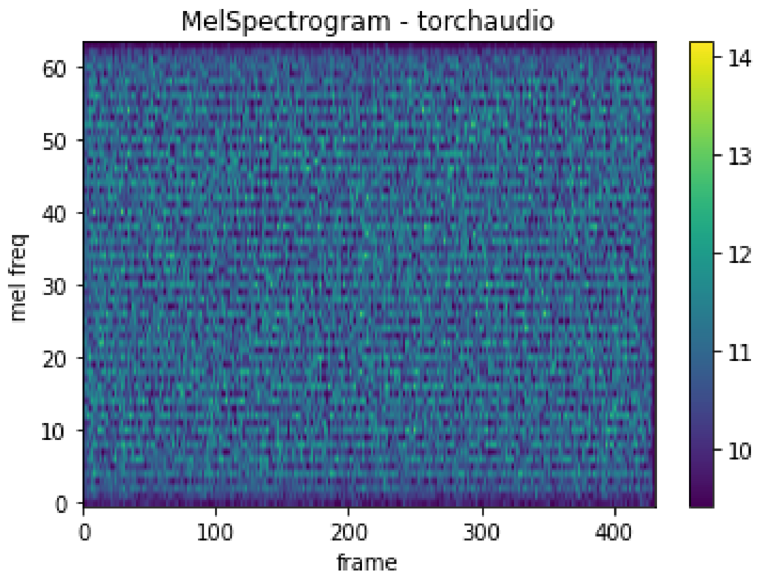


Figure 10: Test data output after forward pass through trained VAE model

As shown above in Fig.10, when using the same sample input from the testing data (this time with a masked end instead), the VAE-CNN model did not output any learned features, and instead outputted an array of noise. Perhaps this is because the mel-spectrogram was too complex, and the model architecture too small, for any distinguishable features to be learnt. As there are related works to the use of mel-spectrograms and Convolutional VAEs [], it was surprising to see that no features had been learnt. After experimentation with l1norm loss, it was found that the model was learning, but learning slowly. There was no sign of overfitting, perhaps because the data had been appropriately normalised. However if the model had been trained for more epochs, as the data was so complex, there might have been more signs of the model learning the mel-spectrograms features.

6 Discussion

Both models could have been improved by adding batch normalisation layers after each convolutional layers, and max pooling after one or two convolutional layers. They might have benefited from data that normalised in a different way, or the denormalisation of the data during inference might have affected the output of the testing data sample from the trained models. Some audio data would have also been lost in the transformation to the mel-spectrogram, as well as its transformation back to audio through the pytorch transformation. As both models were small networks, many features would have been lost in training. As the loss function was still reducing for the seconf module, it indicated that the model could have trained for more epochs, and might have then picked up more audio features. The size of the masking would have also had an effect on the amount of features that the model could learn. Unless the models contained many more trainable parameters, the reconstruction of the audio in inference would be considerably poor.

7 Future Work

Possible considerations for future work could include the exploration of transformer models for these tasks. Recent works have used quantised audio data that has been transformed into MIDI data, and tokenised before being passed through a transformer model[4]. Compound word transformed demonstrated that the 'masked end' of audio data could definitely be predicted (although as a continuation of audio data when given a primer input), and it will now be interesting to see if this method could be explored for the task of audio infilling too.

References

[1] L. Wyse. Audio Spectrogram Representations for Processing with Convolutional Neural Networks. 1(1):37–41, 2017.

[2] Théis Bazin, Gaëtan Hadjeres, Philippe Esling, and Mikhail Malt. Spectrogram Inpainting for Interactive Generation of Instrument Sounds. pages 1–16, 2021.

[3] Meinard Müller, Verena Konz, Wolfgang Bogler, and Vlora Arifi-Müller. Saarland music data (SMD). 2011.

[4] Wen-Yi Hsiao, Jen-Yu Liu, Yin-Cheng Yeh, and Yi-Hsuan Yang. Compound Word Transformer: Learning to Compose Full-Song Music over Dynamic Directed Hypergraphs. 2021.