Music Generation

Research style project:
An exploration of Melspectrogram.

cMelGAN - conditional generative model based on MelSpectrograms, a faster model for limited hardware computing resources

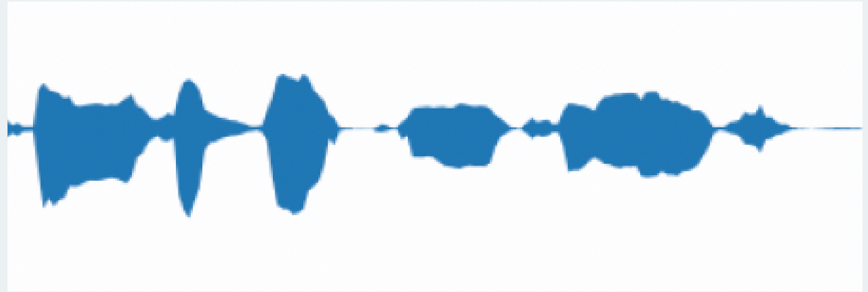
ECE324 UNIVERSITY OF TORONTO

APRIL 2022

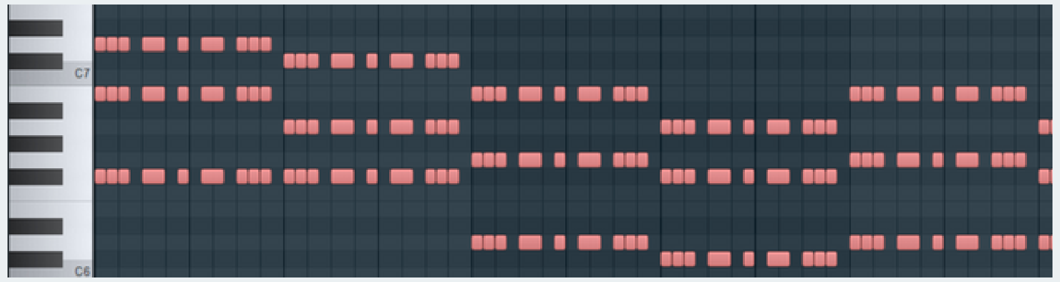
Digital Representation of Music

• Sound is a **wave**	• Sheet music of **notes**	• Present music in **time** and **frequency** domain • Closest to **human perception** of music

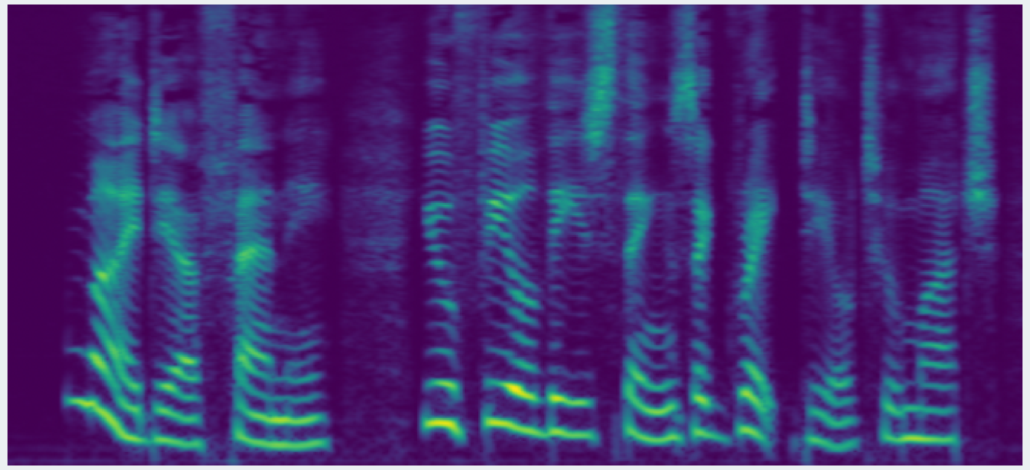
WAV form



MIDI

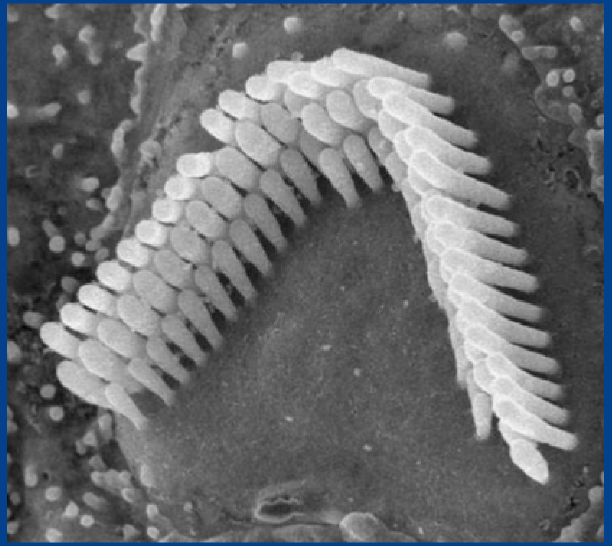


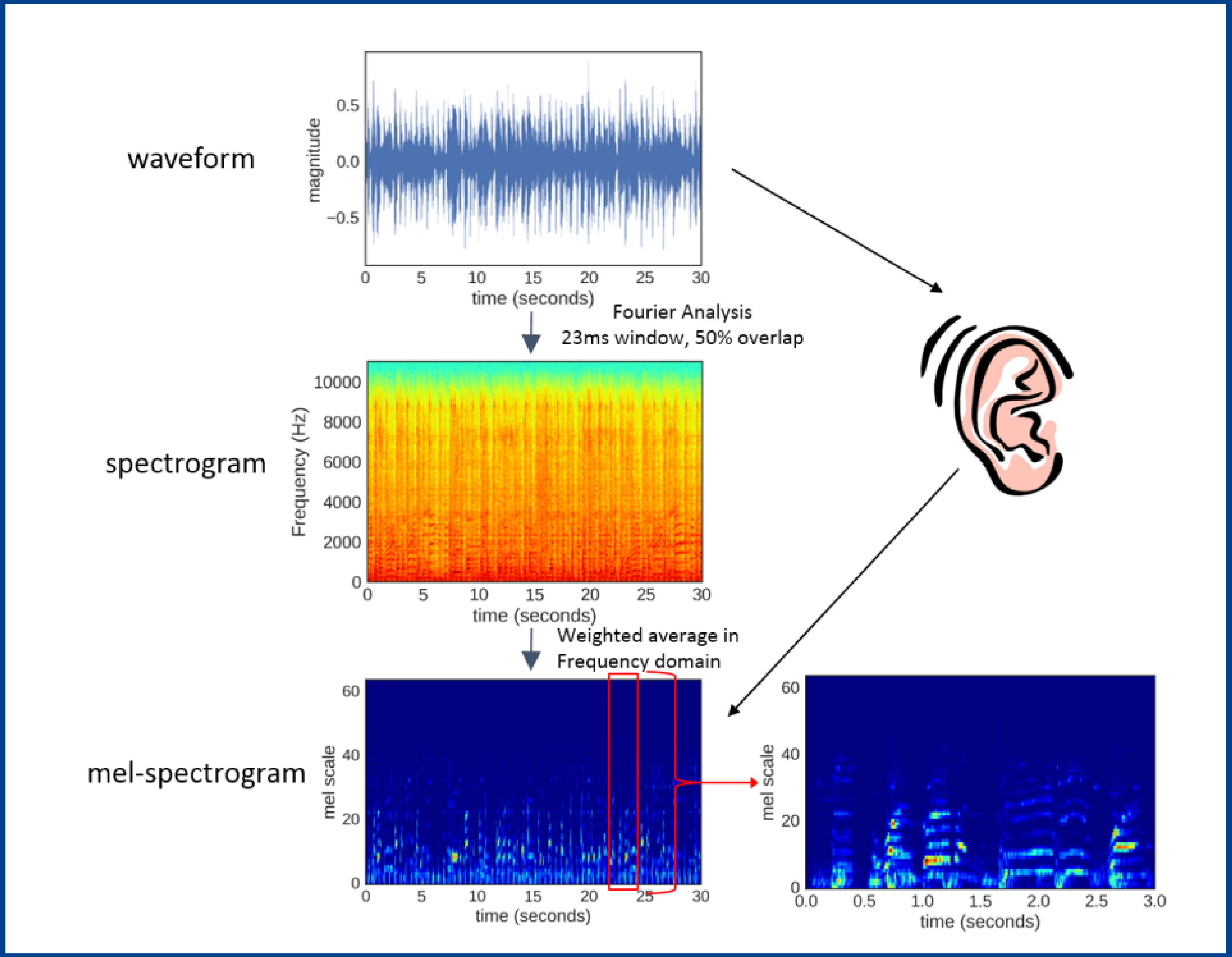
MelSpectrogram



Mel Spectrogram

pictures of sounds





- 2-D representation of audios by taking **fourier transform** of the waveform at regular interval.
- Human cochlea contains hairs associated with **one particular frequency** to extract the frequency information given an audio **wave**.
- Would this representation generate **better** music?

Fourier analysis happening right now...

Problem Statement

PROBLEM STATEMENT

PASTWORK

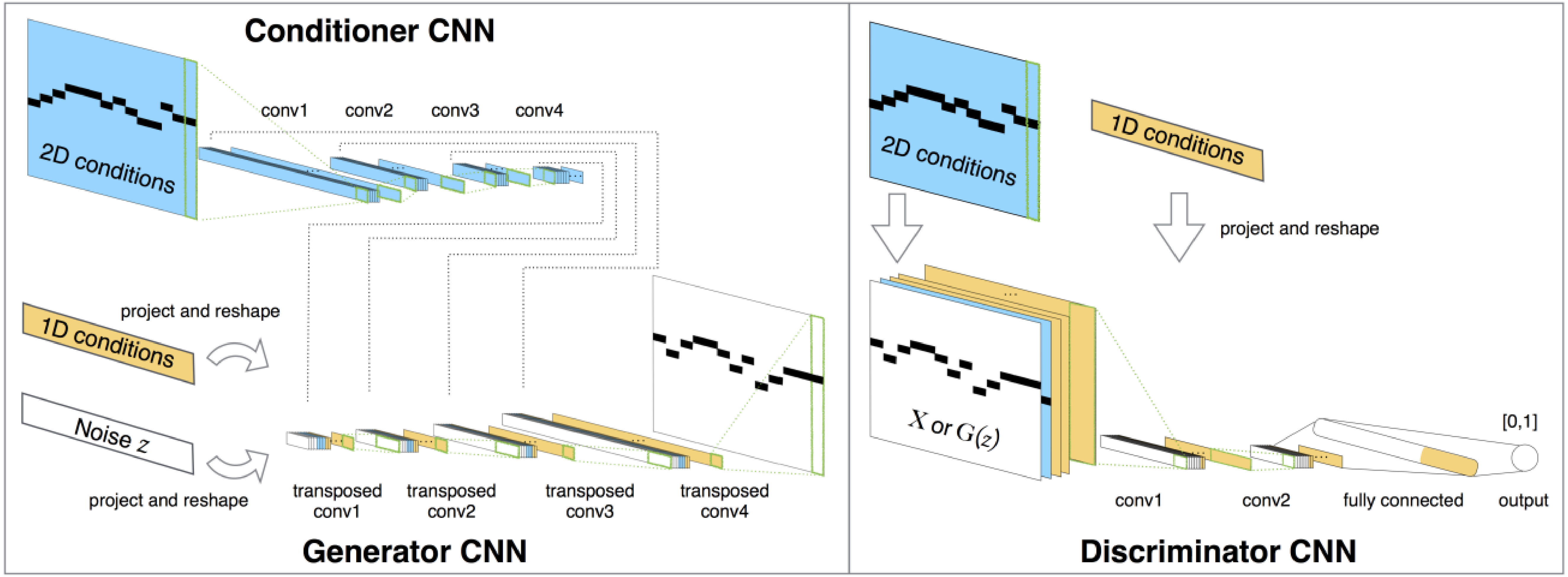
DATACOLLECTION

- Can Mel Spectrograms **outperform** traditional MIDI-based formats?

- Add genre **conditioning** because its **rarely** been done in generative models based on MelSpectrograms
- Potentially **closer** to human perception
- Downside is **less structured** format so harder for network to learn good representations
- Maybe similar to how convolutional networks match human perception of images and so do better on image-based tasks

PAST WORK

MidiNET:
Convolutional GAN Music Generator



HTTPS://ARXIV.ORG/PDF/1703.10847V2.PDF

- h = # of MIDI Notes
- w = # of time steps
- Noise vector z gets fed into the Generator, G.
- Each entry of G(z) gets simultaneously conditioned by 2D conditions (Blue Block)
- Output of G(z) is h x w matrix.

Data Collection

Data Collection

- **3** Genre Total **18** hours of music
- youtube-dl to scrape music based on genre
- Subset of MAESTRO dataset
- torchaudio to resample and automatically generate melspectrograms from audio, use network to predict next frame of spectrogram
- Allows for **conditional generation** based on genre

Data Pipeline

Scalable **custom MusicDataset** pipeline using Pytorch Dataset class
Divide music into training samples on the scale of **seconds**
Convert it from **wave form** to **MelSpectrogram**

MelNET

Autoregressive element-by-element music generation

- **RNN** based autoregressive conditional generative model
- **Time-delayed** stack and **Frequency-delayed** stack
- Feature **extraction** networks
- **Gaussian** mixture modelling
- **Multiscale** modelling

$$p(x) = \prod_i \prod_j p(x_{ij} \mid x_{<ij}; \theta_{ij})$$

$$p(y|x) = \sum_k \pi_k(x)\, N(y|\mu_k(x),\ I\sigma_k^2(x)).$$

- **Training objective:** minimizing mixture density networks loss

$$L(w) = \frac{-1}{N} \sum_{n=1}^{N} \log\left(\sum_k \pi_k(x_n, w) N(y_n|\mu_k(x_n, w), I\sigma_k^2(x_n, w))\right),$$

MelNET

Implemented in **Pytorch**

RESEARCH

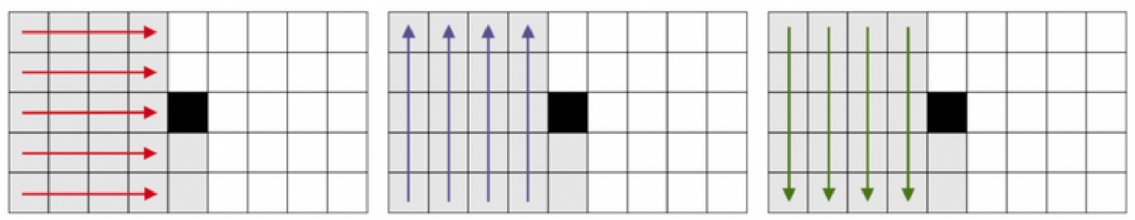
SET UP MODEL

OPTIMIZATION

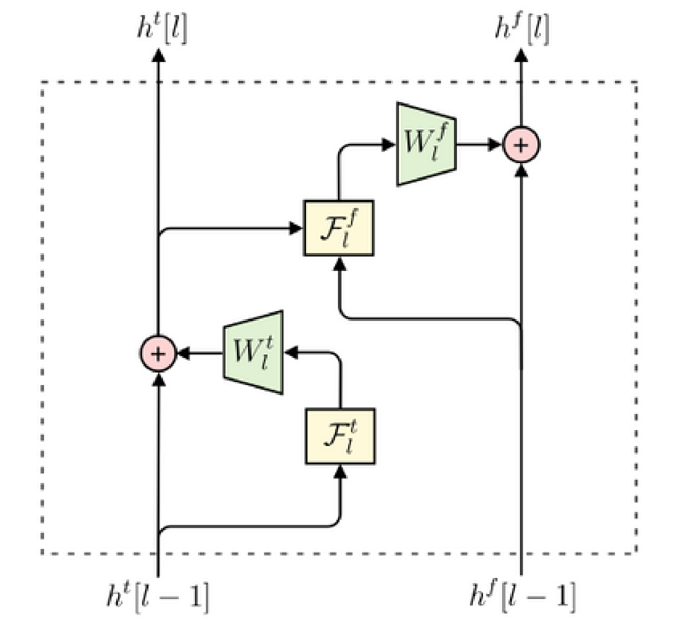
DIVERGE TRAINING

ANALYZE RESUTLS

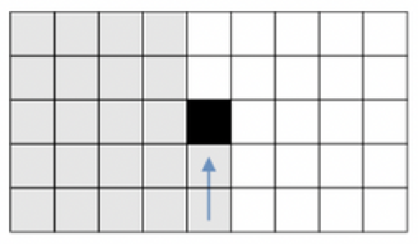
```python
class TimeDelayedStack(nn.Module):
    def __init__(self, dims):
        super().__init__()
        self.bi_freq_rnn = nn.GRU(dims, dims, batch_first=True, bidirectional=True)
        self.time_rnn = nn.GRU(dims, dims, batch_first=True)

    def forward(self, x_time):

        # Batch, Timesteps, Mels, Dims
        B, T, M, D = x_time.size()

        # Collapse the first two axes
        time_input = x_time.transpose(1, 2).contiguous().view(-1, T, D)
        freq_input = x_time.view(-1, M, D)

        # Run through the rnns
        x_1, _ = self.time_rnn(time_input)
        x_2_and_3, _ = self.bi_freq_rnn(freq_input)

        # Reshape the first two axes back to original
        x_1 = x_1.view(B, M, T, D).transpose(1, 2)
        x_2_and_3 = x_2_and_3.view(B, T, M, 2 * D)

        # And concatenate for output
        x_time = torch.cat([x_1, x_2_and_3], dim=3)
        return x_time
```



(a) Time-delayed stack




```python
class FrequencyDelayedStack(nn.Module):
    def __init__(self, dims):
        super().__init__()
        self.rnn = nn.GRU(dims, dims, batch_first=True)

    def forward(self, x_time, x_freq):
        # sum the inputs
        x = x_time + x_freq

        # Batch, Timesteps, Mels, Dims
        B, T, M, D = x.size()
        # collapse the first two axes
        x = x.view(-1, M, D)

        # Through the RNN
        x, _ = self.rnn(x)
        return x.view(B, T, M, D)
```

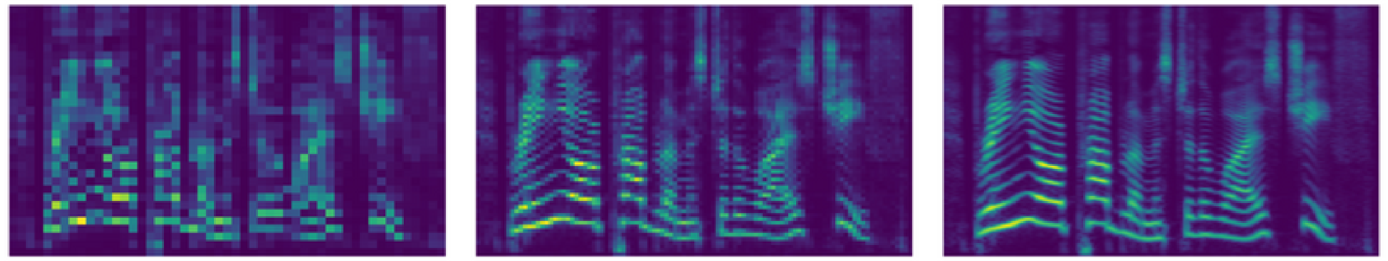


(b) Frequency-delayed stack

$$h_{ij}^{t}[l] = W_l^t F_l^t(h^t[l-1])_{ij} + h_{ij}^{t}[l-1] .$$

MelNET

MULTISCALE MODELLING



(a) Tier 1 (32 × 50) (b) Tiers 1–3 (64 × 100) (c) Tiers 1–6 (256 × 200)

$$p(x;\psi) = \prod_g p(x^g \mid x^{<g}; \psi^g).$$

GENERATES SPECTROGRAMS FROM **HIGH-LEVEL STRUCTURE** TO **FINE-GRAINED** DETAILS BY ITERATING THROUGH THE **MULTIPLE TIERS** THAT A SPECTROGRAM IS PRE-PARTITIONED INTO

EACH TIER CONTAINING HIGHER RESOLUTION OF INFORMATION THAN THE PREVIOUS ONE

FEATURE EXTRACTION NETWORKS

```python
class FeatureExtraction(nn.Module):
    def __init__(self, num_mels, n_layers):
        super().__init__()
        # Input layers
        #self.freq_fwd = nn.GRU(time_steps, time_steps, batch_first=True)
        #self.freq_back = nn.GRU(time_steps, time_steps, batch_first=True)
        self.time_fwd = nn.GRU(num_mels, num_mels, batch_first=True)
        self.time_back = nn.GRU(num_mels, num_mels, batch_first=True)
        self.weights = nn.Linear(2,1)
        #self.num_params()

    def forward(self, spectrogram):
        # Shift the inputs left for time-delay inputs
        # spectrogram: (batch_size, time, freq)
        #print("spec", spectrogram.shape)
        #N,T,F = spectrogram.size()
        #freq_input = spectrogram.transpose(1, 2).contiguous()
        #print("the fatuers extracting from", type(spectrogram), len(spectrogram))
        time_fwd_feats, _ = self.time_fwd(spectrogram)
        time_back_feats, _ = self.time_back(spectrogram.flip(1))
        #freq_fwd_feats, _ = self.freq_fwd(freq_input)
        #freq_back_feats, _ = self.freq_back(freq_input.flip(2))

        #freq_features = freq_features.transpose(1,2).contiguous().view(-1, T, 2*F)
        stacked = torch.stack((time_fwd_feats, time_back_feats), dim=-1)#, freq_fwd_fe
        #assert not torch.any(torch.isinf(spectrogram))
        #print(spectrogram.max())
        #assert not torch.any(torch.isinf(time_fwd_feats))
        #assert not torch.any(torch.isinf(stacked))
        #assert not torch.any(torch.isinf(self.weights(stacked)))
        return self.weights(stacked).squeeze(-1)

    def num_params(self):
        parameters = filter(lambda p: p.requires_grad, self.parameters())
        parameters = sum([np.prod(p.size()) for p in parameters]) / 1_000_000
        print('Trainable Parametersextrac: %.3fM' % parameters)
```

MelNET

Initial approach was **far too slow**
Speed up training by a factor of **10**

- **Improved** resampling processing and MelSpectrogram conversion
- Further reduced disk accesses from **O(N) to O(1)**
- Used a different **batching method** to expand usable dataset
- Set multiple worker, it is now **multi-threaded** using the dataloader API

- Added buffer checkpointing, **~50 times** increase in model expressive power
- Changed the data pipeline to be optimal on Google Colab

MelNET

RESEARCH — SET UP MODEL — OPTIMIZATION — DIVERGE TRAINING — ANALYZE RESUTLS

Model 1 (big)

50.8 M Parameters
~ 5hrs / epoch (3.6 Hz)
30 epochs
DataConfig:
batch_sz=6, num_mels=180,
win_sz=3,
stft_hop_sz=800,
stft_win_sz=256*8

TrainConfig:
dims=256, n_layers=[12,6,5,4],
directions=[2,1]

Model 2 (small)

4.04 M Parameters
~ 1hr / epoch (18 Hz)
30 epochs
DataConfig:
batch_sz=6, num_mels=180,
win_sz=8,
stft_hop_sz=800,
stft_win_sz=256*8

TrainConfig:
dims=64, n_layers=[12,6,5,4],
directions=[2,1]

MelNET

Very hard to train, need to build another model.

Comparison results at the end of the presentation.

FIRST OF EVER
WE PROPOSE:

CMelGAN

Inspired by MelGAN and cGAN
Conditional Generative Adversarial Networks
based on Melspectrograms

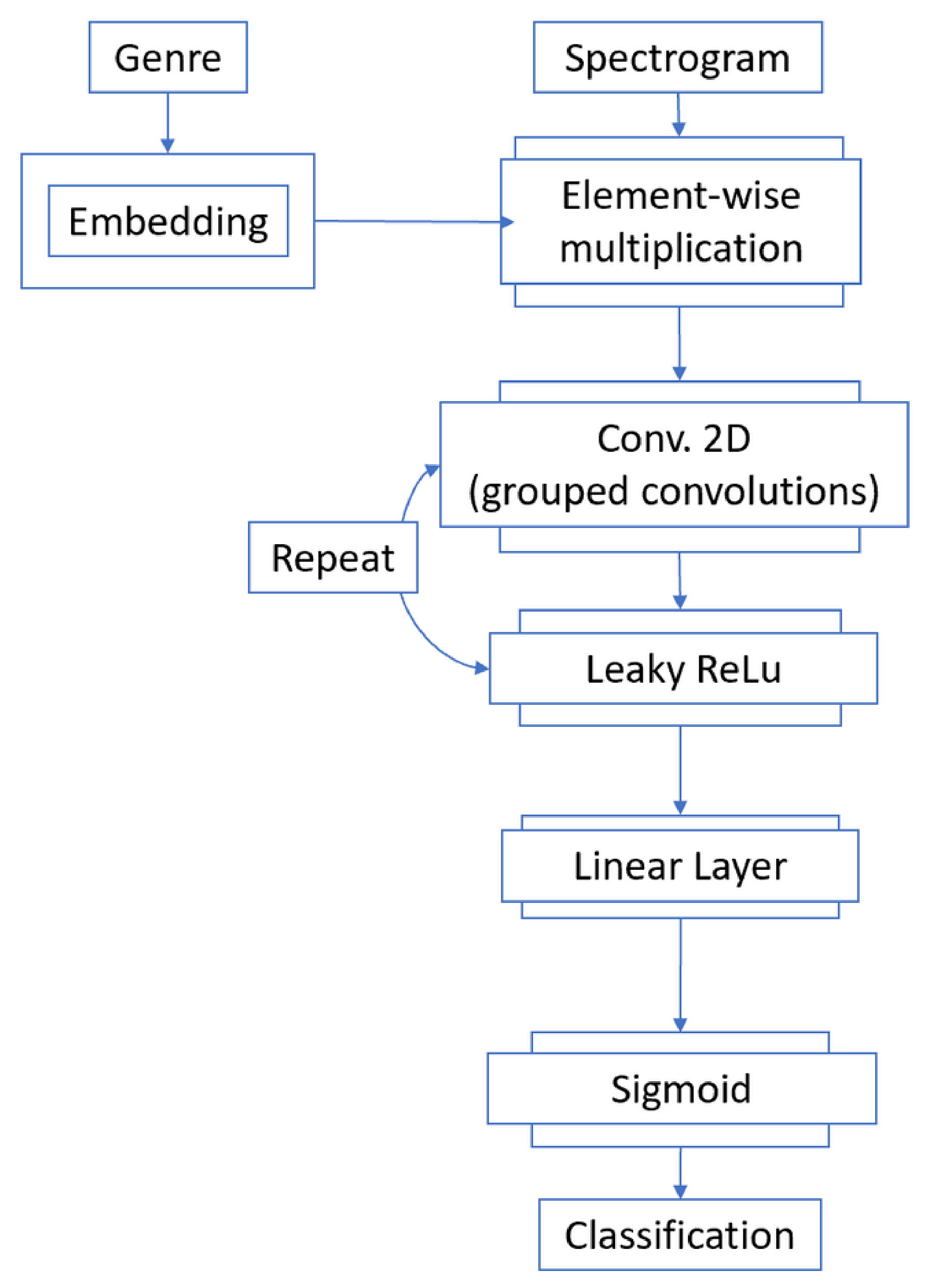
CMelGAN

SET UP MODEL

OPTIMIZATION

ANALYZE RESUTLS

Discriminator



Combines ideas from MelGAN and CGAN

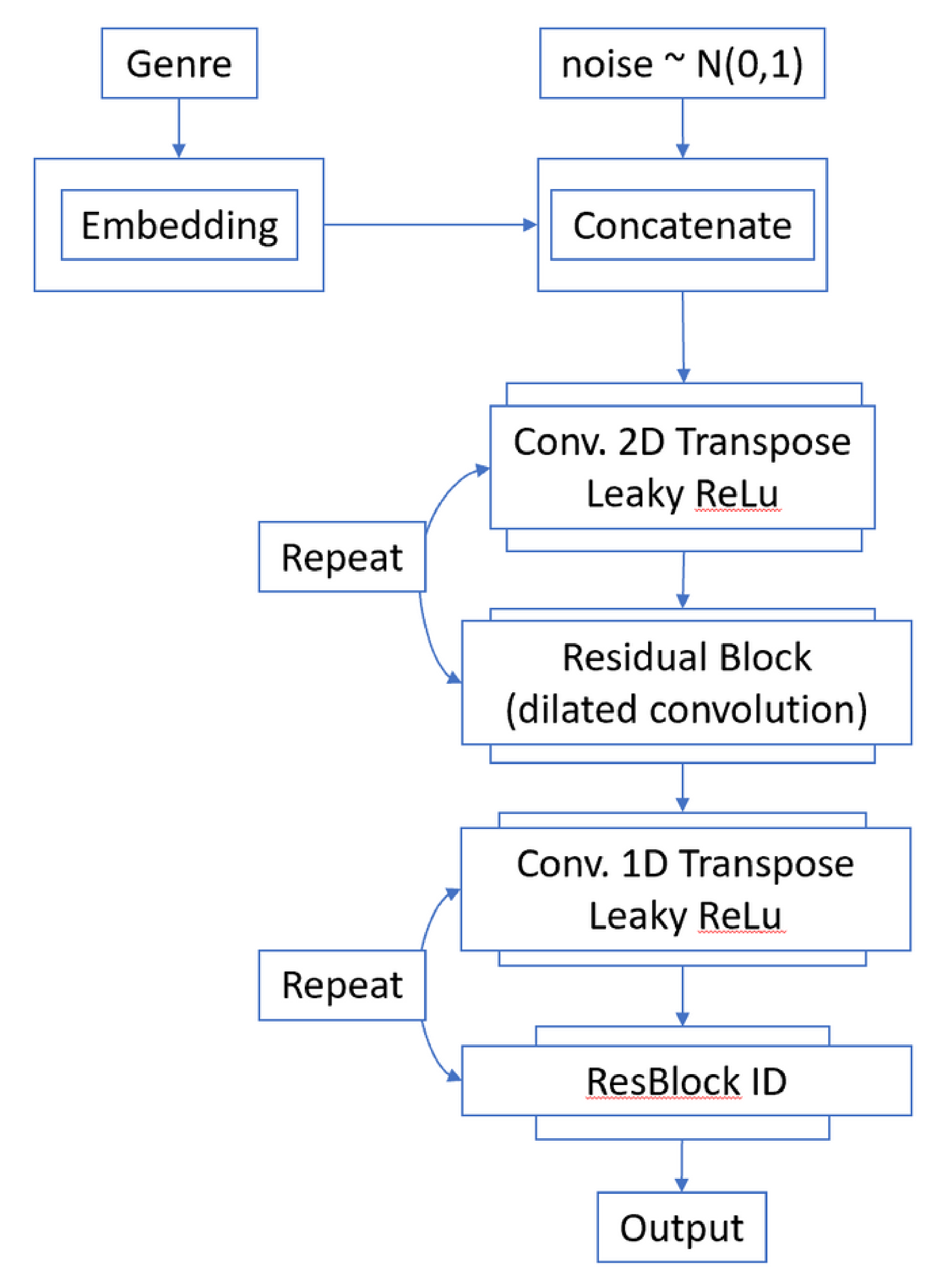
Generator

- **Embedding** layer for genre, elementwise multiplication
- Transposed **2D** convolutions, with weight normalization
- **Dropout** and batchnorm for regularization
- **Residual** blocks with **dilated convolutions**
- Transposed 1D convolutions, with weight normalization
- **Dropout** and 1D batchnorm

Discriminator

- **2D** grouped convolution, linear layer

Generator



CMelGAN

SET UP MODEL

OPTIMIZATION

ANALYZE RESUTLS

Optimization of model

- Run on **tiny fraction** of dataset (~10%) to get quick look at hyperparameter performance
- Most hyperparameter settings were unstable, with either the discriminator either being easily fooled, or the generator not powerful enough to fool the discriminator
- With fractional trick, were able to test far more hyperparameter settings
- Settled on some decent ones that made neither discriminator nor generator dominate
- Add dropout, batchnorm, and weight normalization (bag of tricks)

Preliminary Results

RESEARCH

SET UP MODEL

ANALYZE RESUTLS

Model	MOS (/5)	Processing Rate (train)
MelNet (small)	0.0 (random noise)	307 kHz
MelNet (large)	0.0 (random noise)	75 kHz
GriffinLim (MelNet settings)	0.5 (basically random noise)	-
ConditionalMelGAN (medium)	0.0 (random noise)	980 kHz
CMelGAN (large, no 1D Convolution)	0.0 (random noise)	705 kHz
GriffinLim (CMelGAN settings)	0.5 (basically random noise)	-

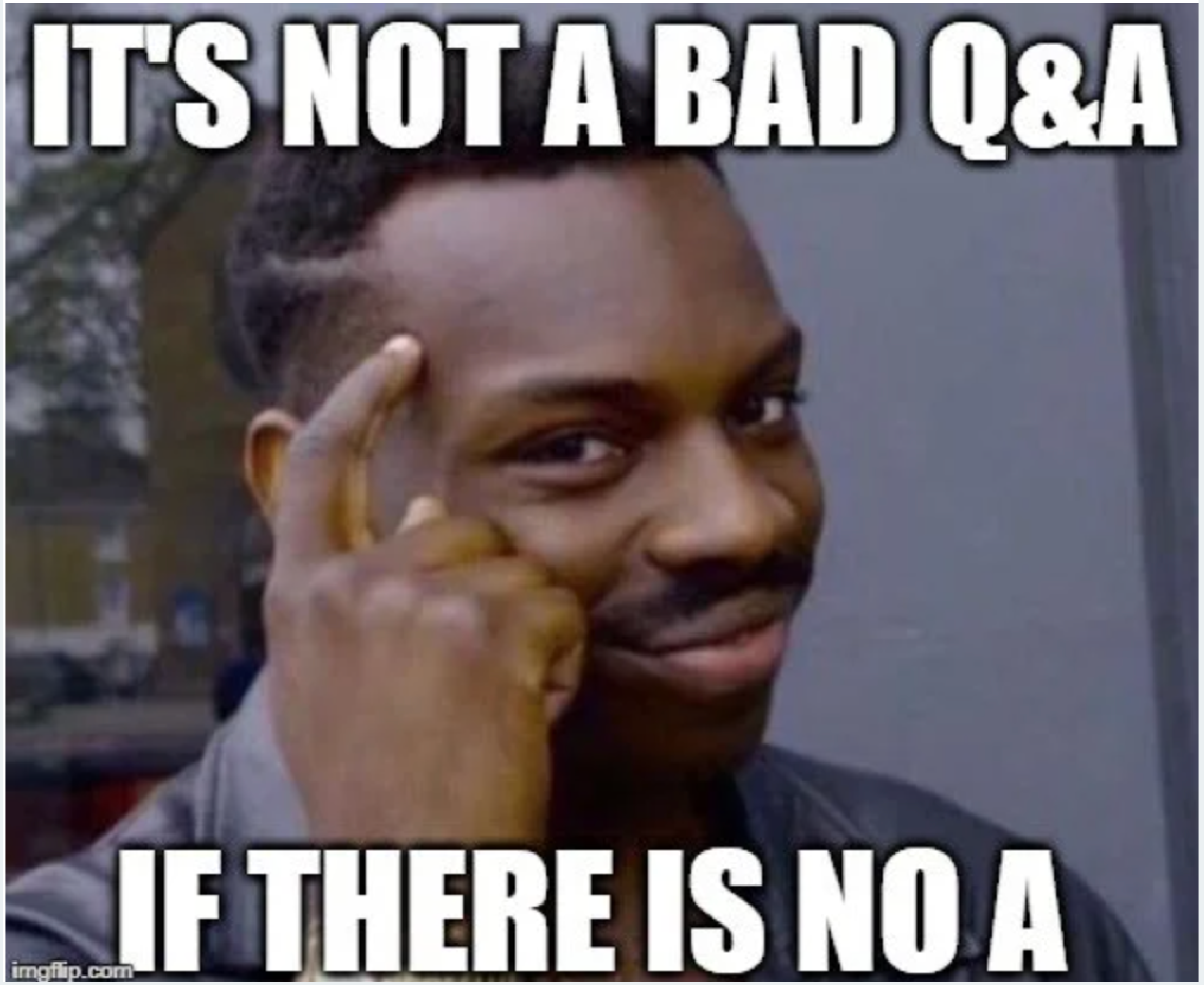
Conclusion and Next Steps

RESEARCH

SET UP MODEL

ANALYZE RESUTLS

- Melspectrograms are really **hard to work with**
- **Unstructured format** makes it too hard for networks to learn a good representation
- **Griffin-Lim** algorithm not good enough for inversion, need to add a neural vocoder
- MIDI approach seems to be **better option**, at least on **limited hardware**
- Network was **unable** to learn a good representation of music
- Probably underestimated the difficulty of the project


IT'S NOT A BAD Q&A
IF THERE IS NO A
imgflip.com