CLAUDIO

Classifying Audio

By Bastian, Frida, Julie & Thea

The *Pitch*

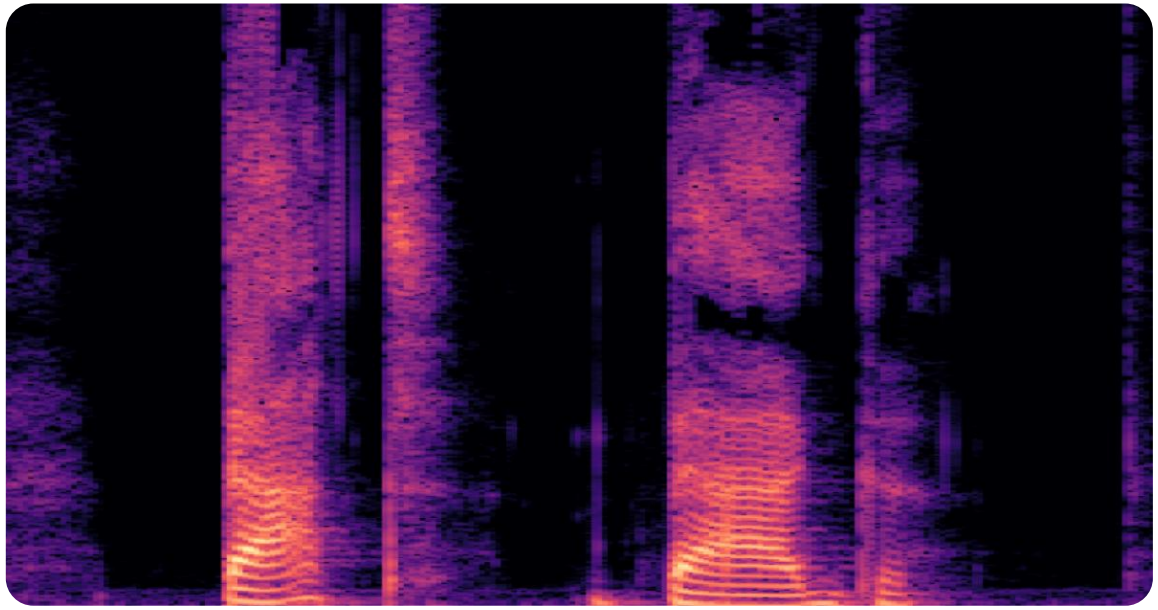
Control your computer with sound

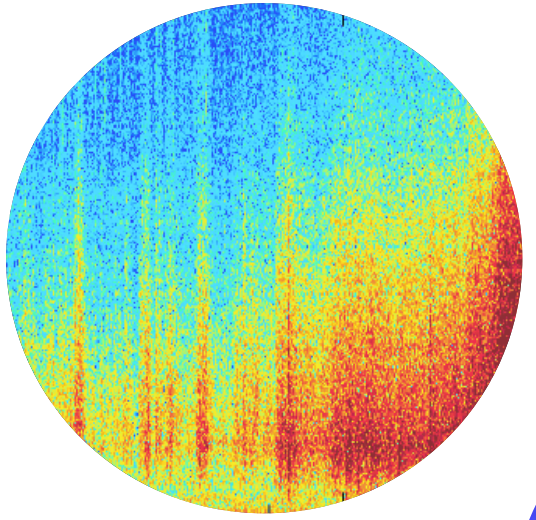
To do that we need to:

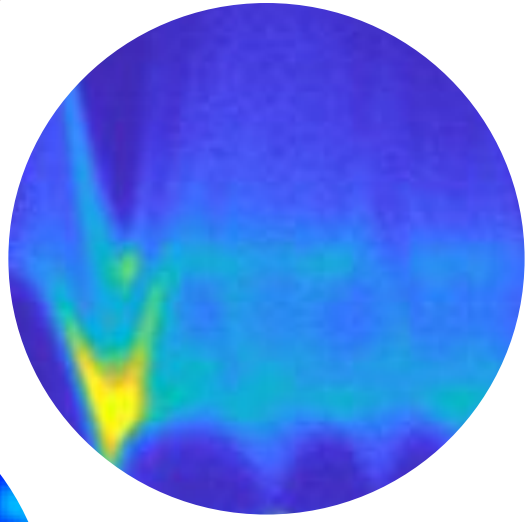
- Recognise and categorise sounds
- Be able to process live audio

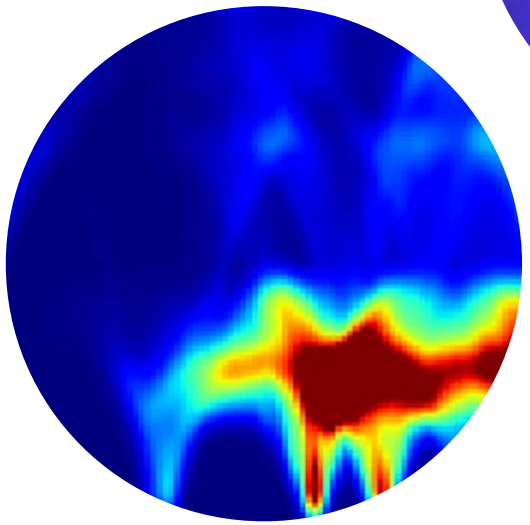
The Motivation

- Analyse Spectrograms as 2D Images
- A useful method for many applications!









What Have We Done?

Data Preparation

- Normalisation
- Precaching
- Data Augmentation

Convolutional Neural Networks

- Basic CNN
- ResNet18
  - Squeeze + Excitation

Other Methods

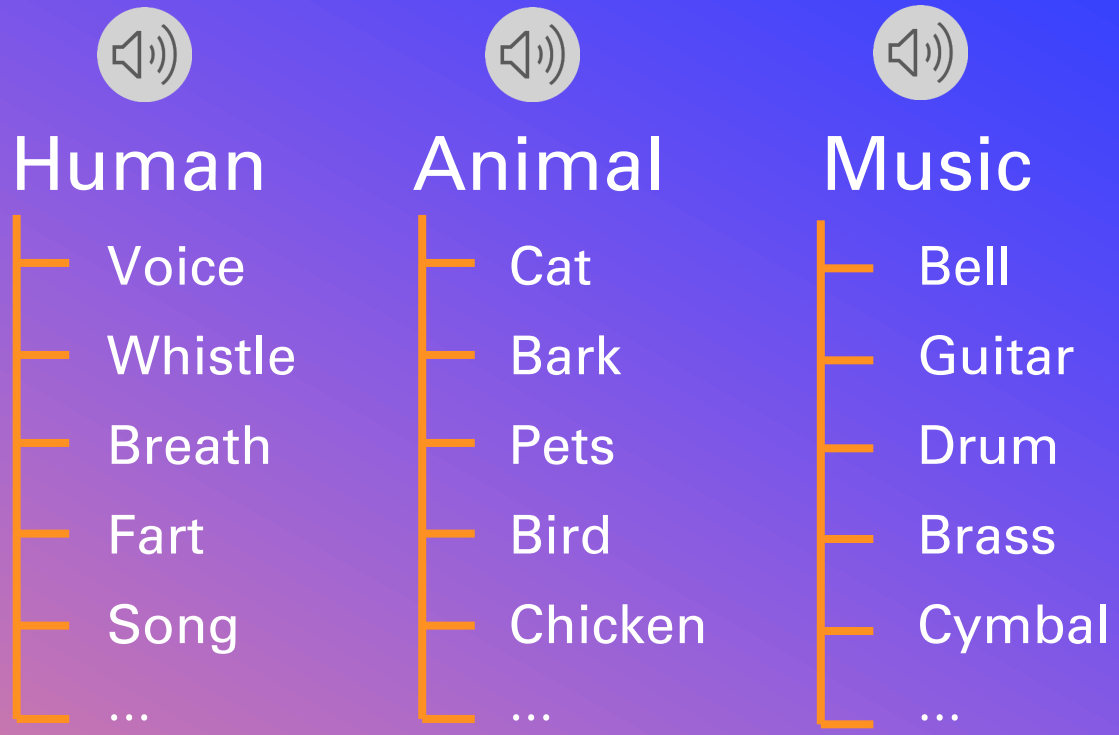
- XGBoost
- Transformer
- Federated Learning
- CLAP

Optimise for GPU

- Implement Data Loaders

The Data – FSD50K

- 51.197 Audio Clips
- 200 labels
- Data splits
  - Develop
  - Evaluate
- Filetype: .wav



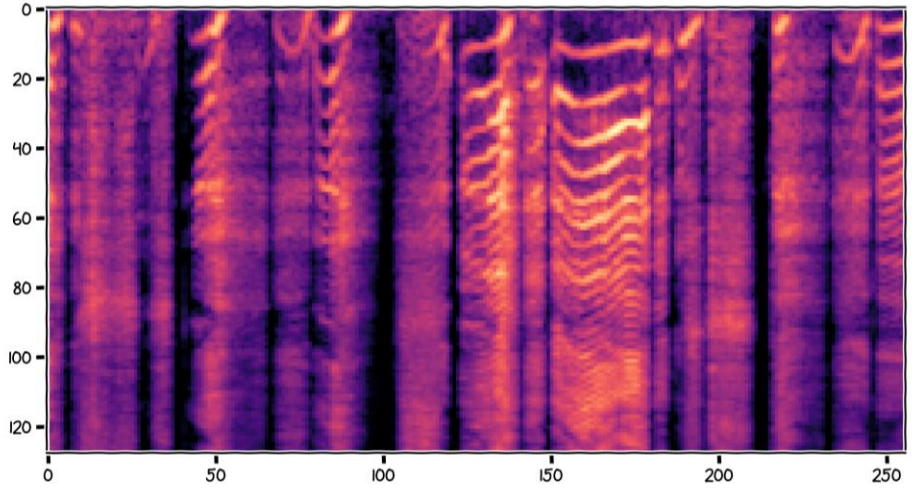
The Data – Preprocessing

Problems:

- Range from 0.3 – 500 s
- Arbitrary intensities
- Full data is ~25 GB

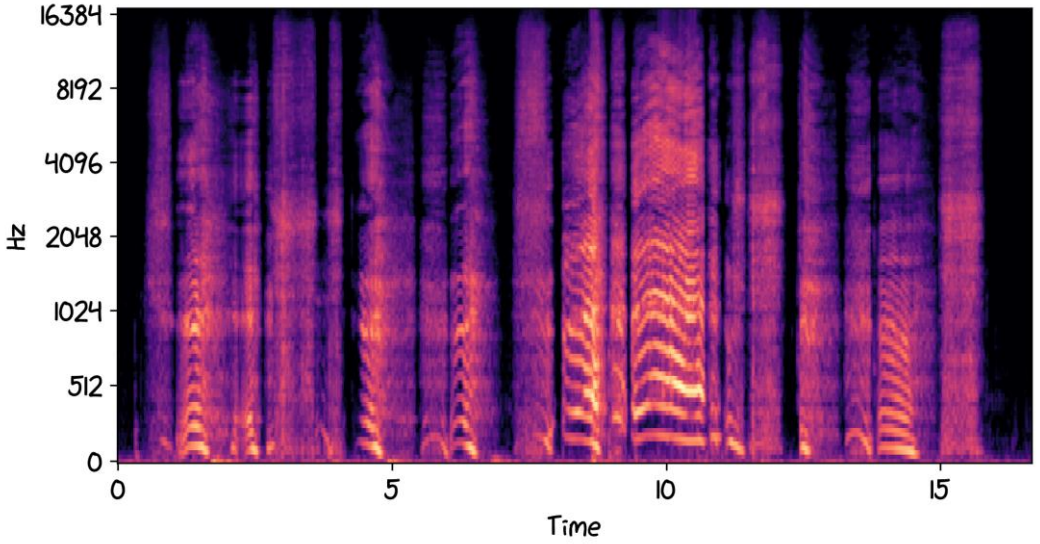
Solutions:

- Repeat/Trim to 5.94 s
- Normalise volume
- Precache tensors

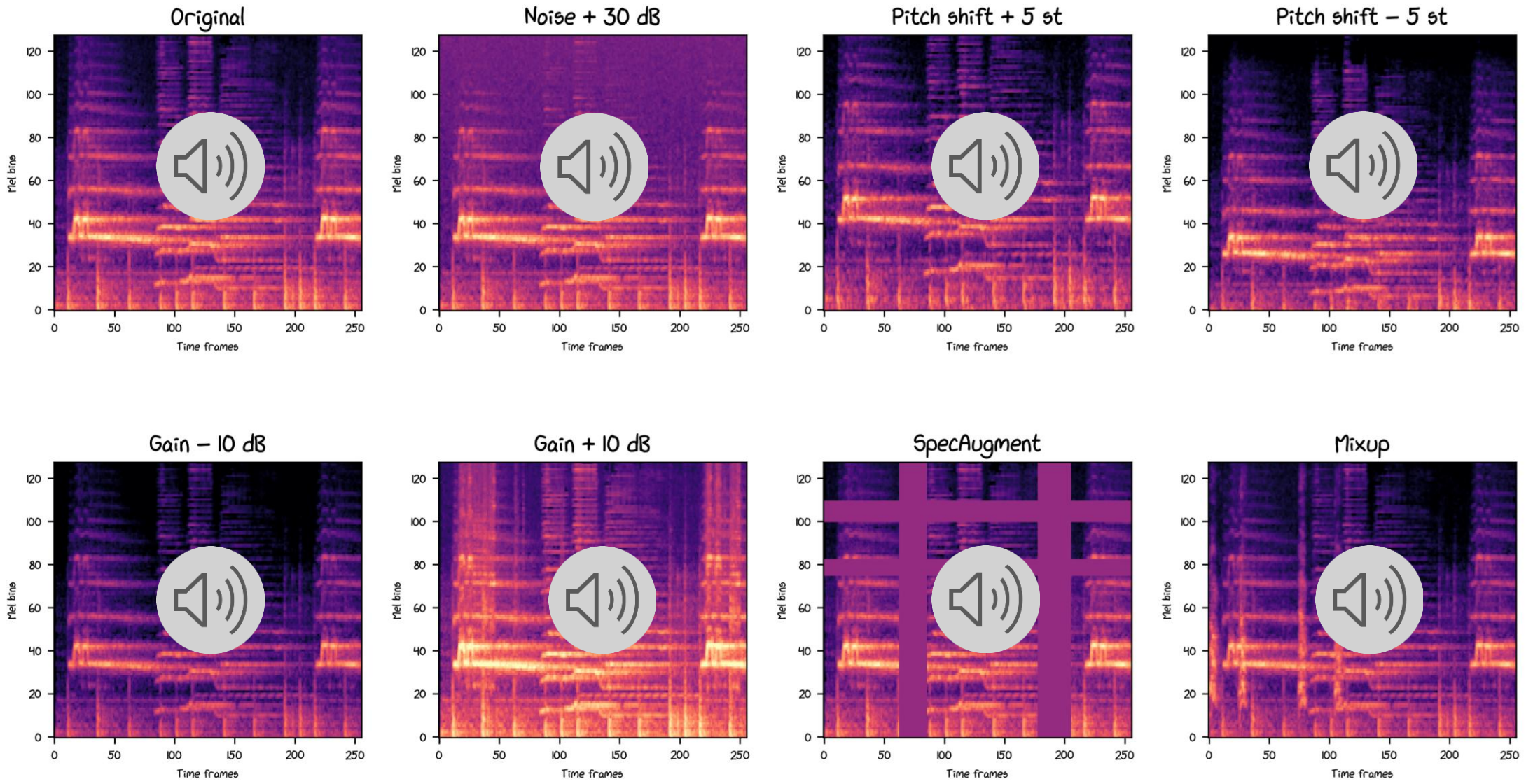


Waveforms to Spectrograms

- Not too big, not to small
  - 64 x 128 → ~ 2 GB
  - 128 x 256 → ~ 7 GB



The Data – Augmentation



Evaluation Metrics

Loss Function:

- Binary Cross Entropy*

***Problem**

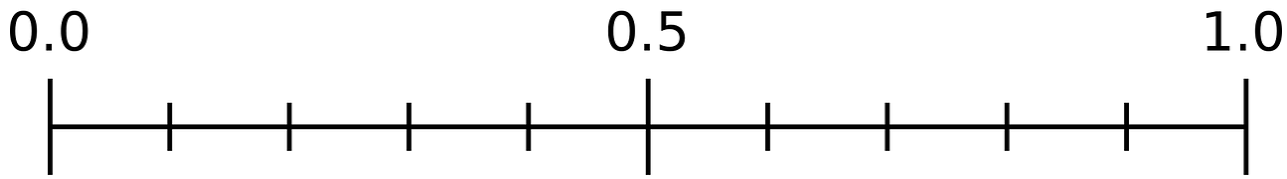
- Clips have 2-3 Labels
  [0,0,1,0,0,…,0,1,0,0,0]
- Good guess is always
  [0,0,0,0,0,0,0,…,0,0,0]

***Solution:**

Positive weights ~ 1 – 10

Accuracy Metrics:

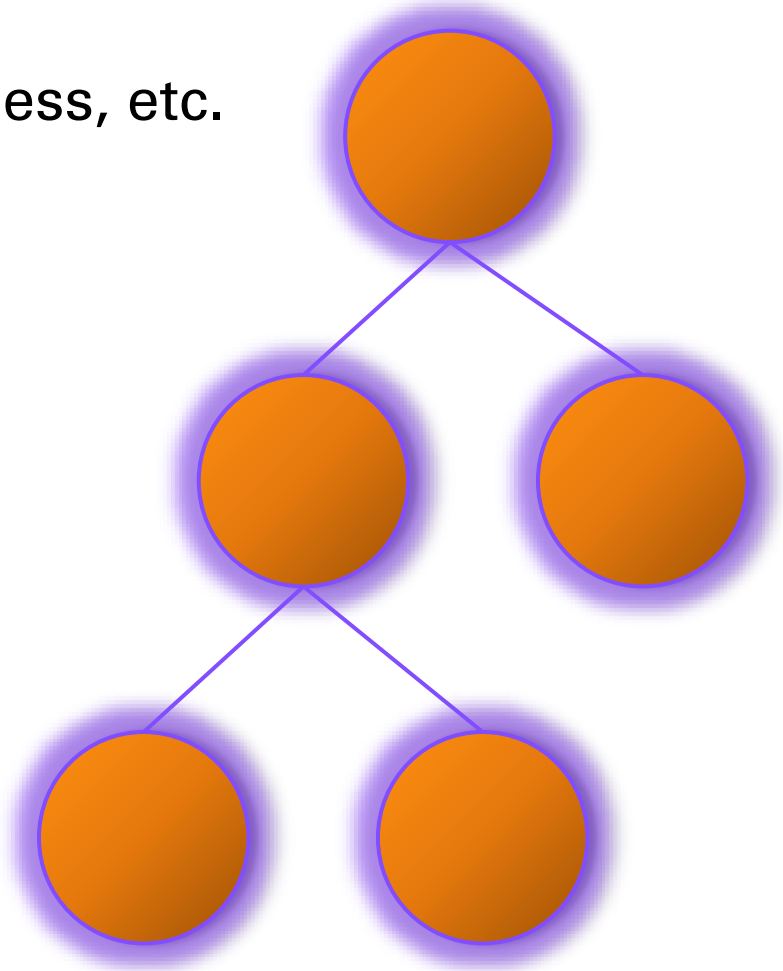
- Precision: *How often is it right?*
- Recall: *How many did it catch?*
- F1: *Mean of Precision & Recall.*
- mAP: *Mean Average Precision.*



MODELS

XGBoost – The Simple Approach

- Extract 36 features such as loudness, noisiness, etc.
- Train on 15 most important features
- Hyper parameter optimisation
- Top performing labels:
  - Applause, Music, Thunder, Human Voice

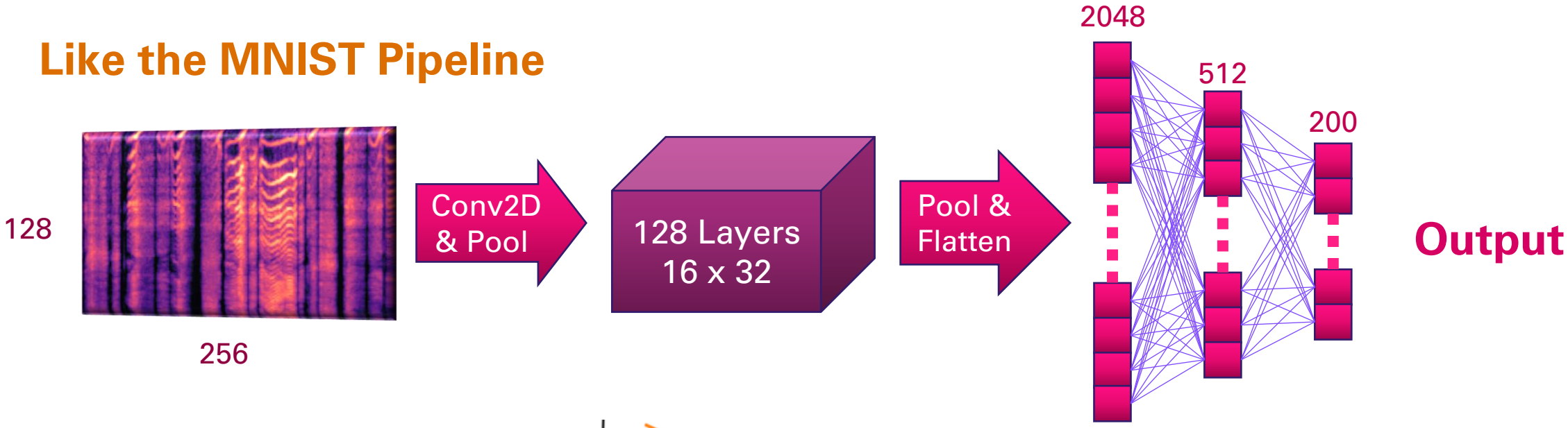


In a few words:

- Number of leafs: ~ 4.000.000
- Best mAP: 0.2837
- Fast and cheap

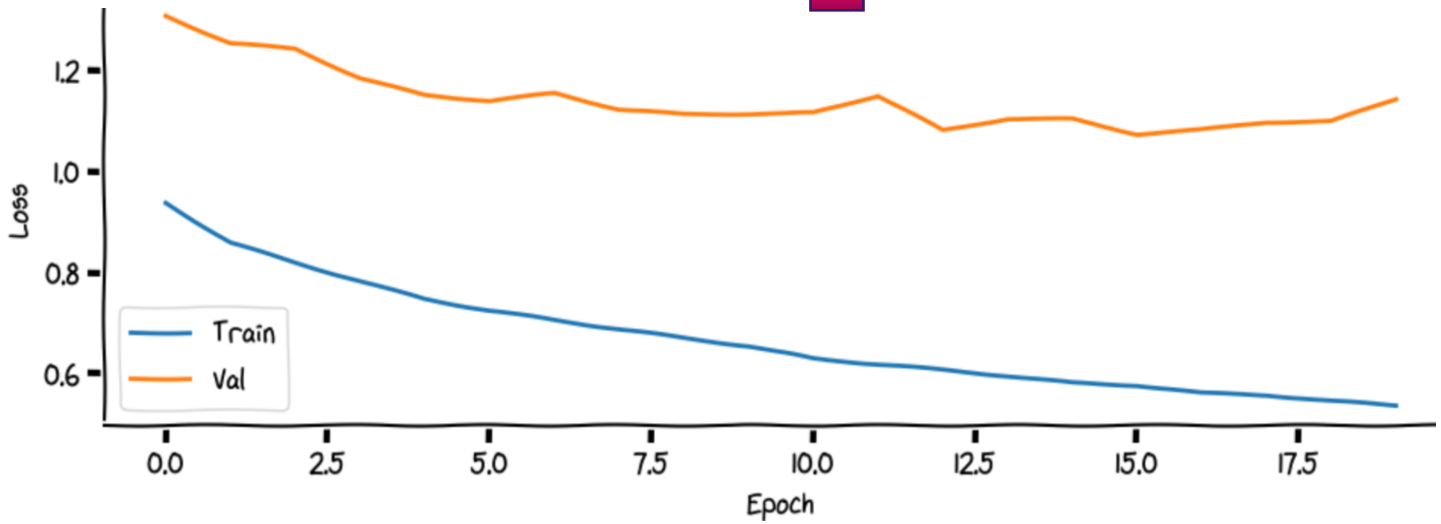
Simple CNN – The Naïve Approach

Like the MNIST Pipeline



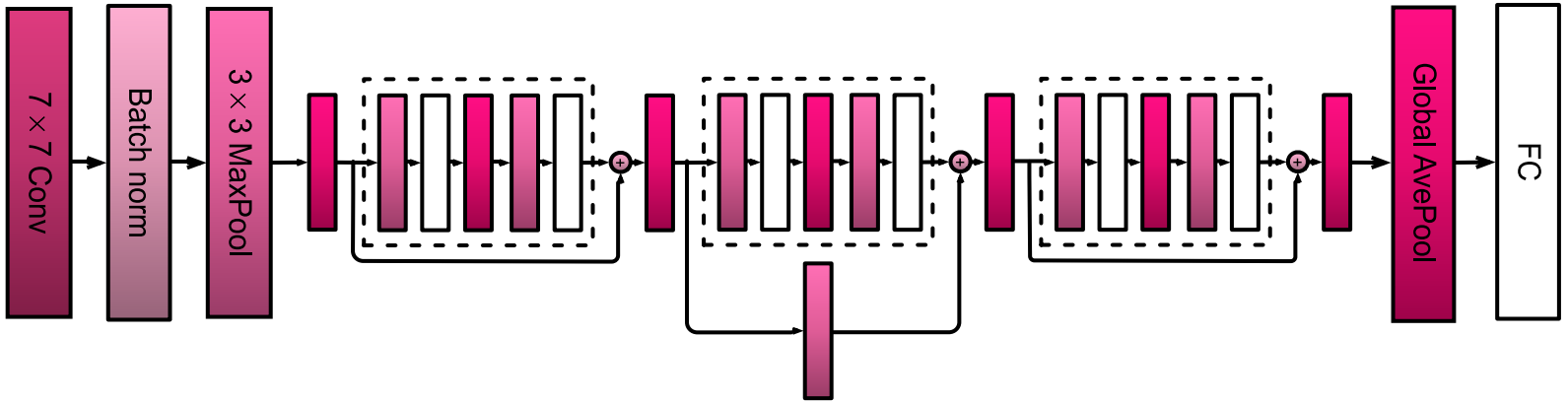
In a few words:

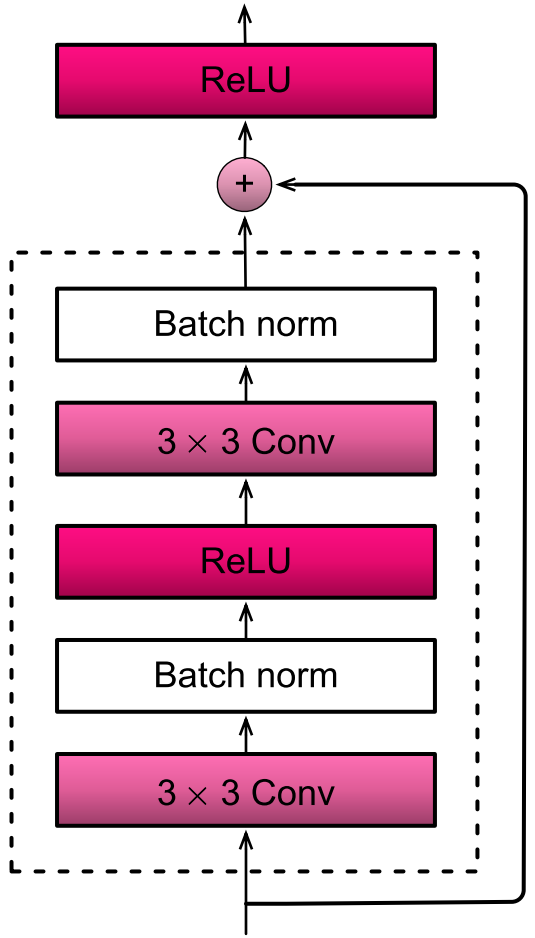
- Parameters: ~ 669.000
- Best mAP: 0.4616
- Prone to overfitting



ResNet18 – The Recursive Approach

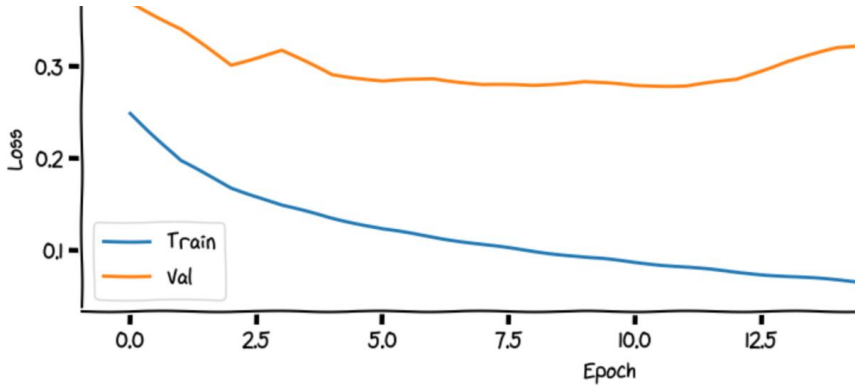
Residual Network





In a few words:

- Parameters: ~11.273.000
- Best mAP: 0.5465
- Also prone to overfitting



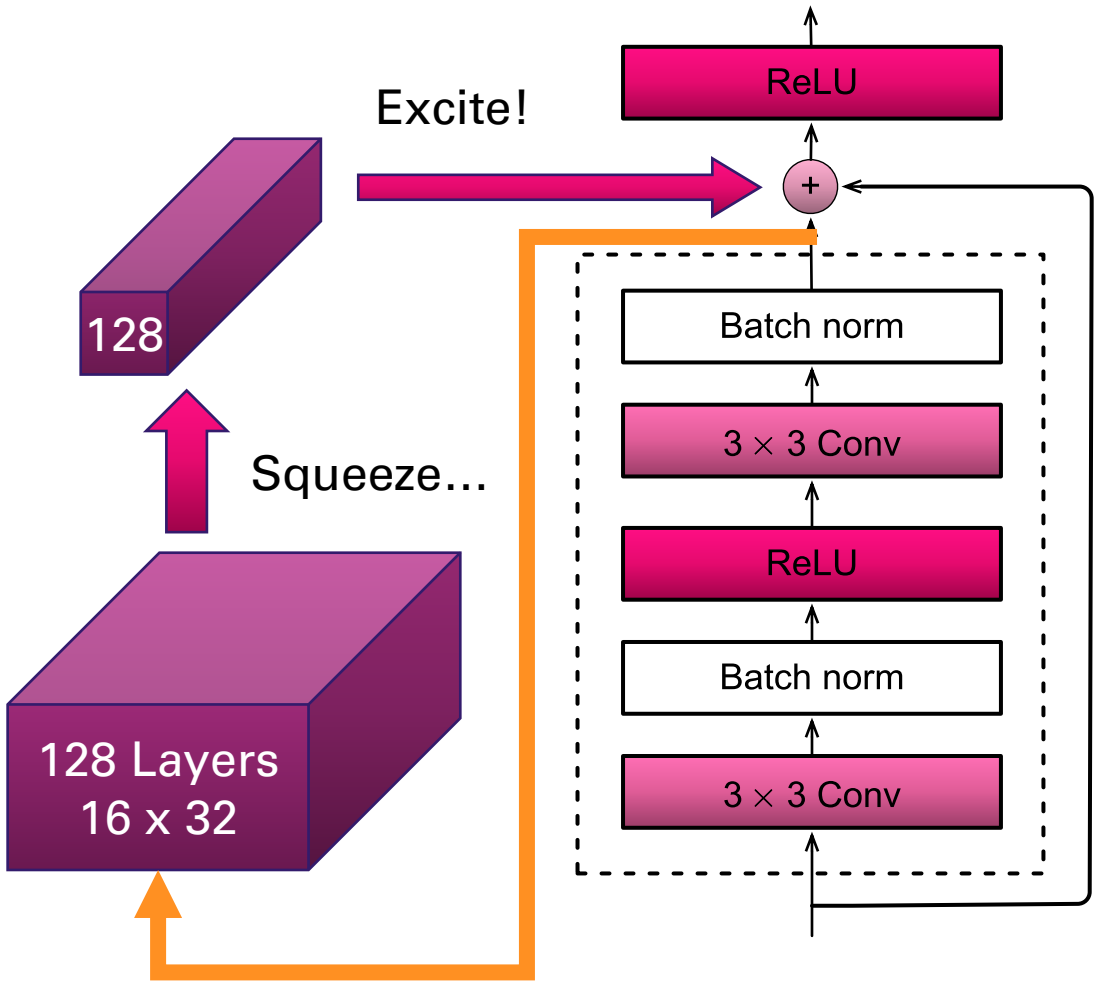
ResNet18 – With Added Attention

Squeeze & Excitation

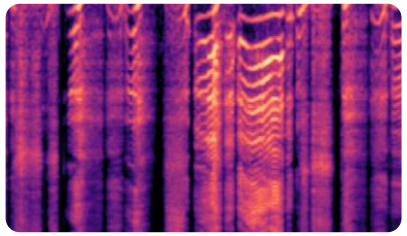
- Focus on important channels
- Collapse each channel to 1 number
- Use this number as a channel weight

In a few words:

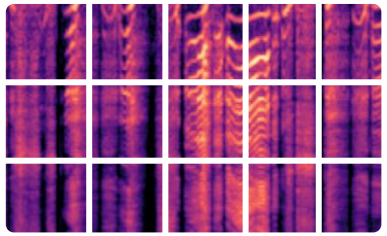
- Parameters: ~ 11.362.000
- Best mAP: 0.5588
- Still prone to overfitting



Transformer – Attention Is All You Need

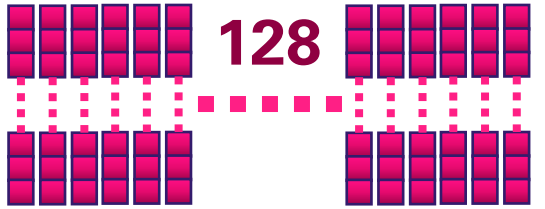


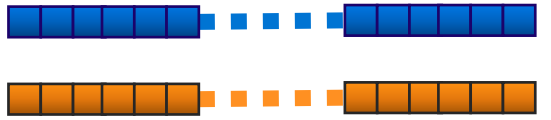
Conv2D



128 Patches
16 x 16

Flatten

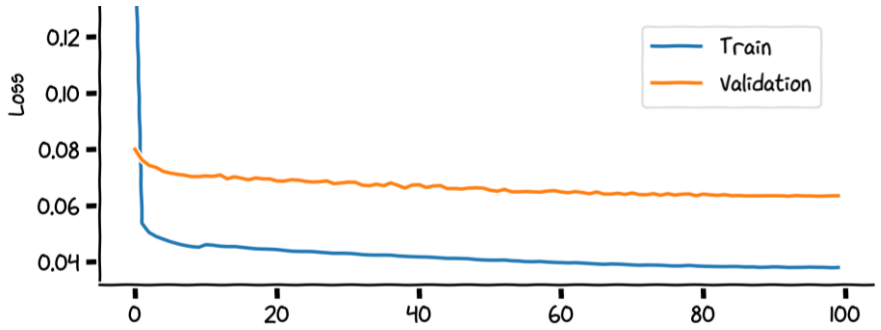




130
Tokens

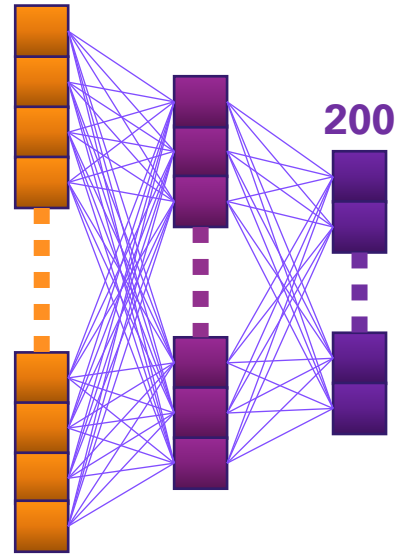
In a few words:

- Parameters: ~ 4.889.000
- Best mAP: 0.4596
- No overfitting (yet)



Transformer Block

CLS



CLS

Output

Federated Learning

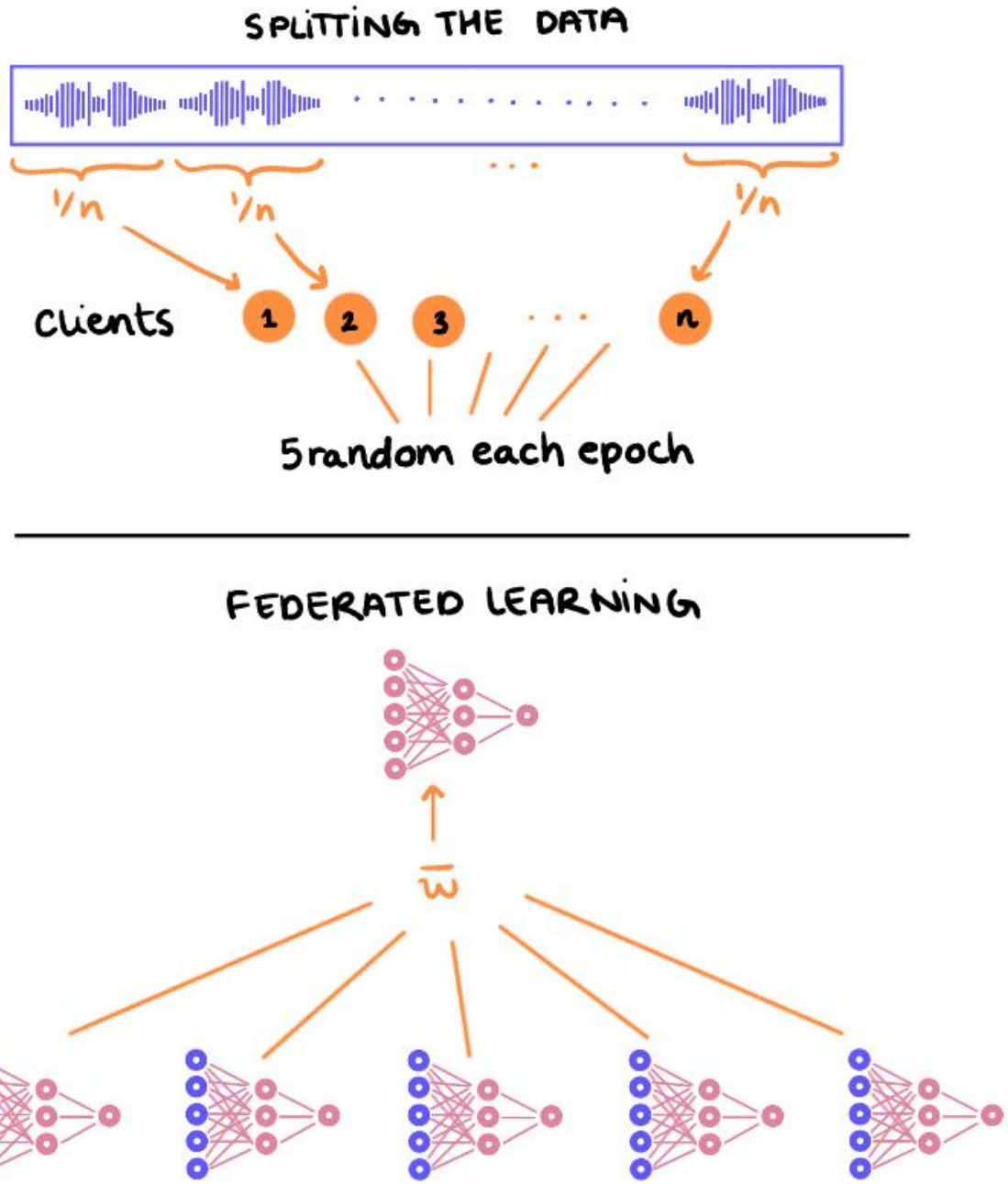
The Setup

Model used: ResNet18

- Rounds = 50
- Clients = 20
- Clients pr round = 10

In a few words:

- Seemed relevant for audio
- Best mAP: ~ 0.18
- Clearly could be improved



CLAP – A Pretrained Solution

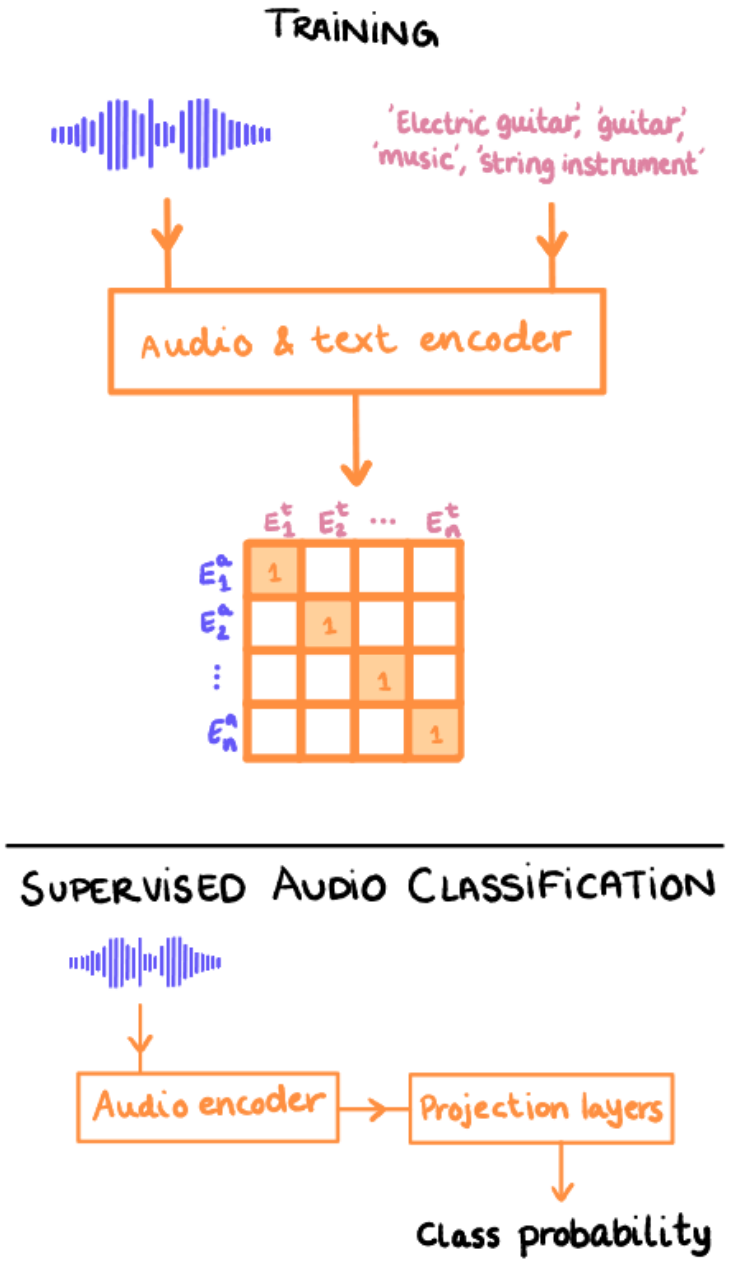
Contrastive Language-Audio Pretraining

- Built in preprocessing!

Sample rate [Hz]	mAP
22050	0.57031
44100	0.57044
48000	0.57042

In a few words:

- Pretty effective
- mAP: ~ 0.57

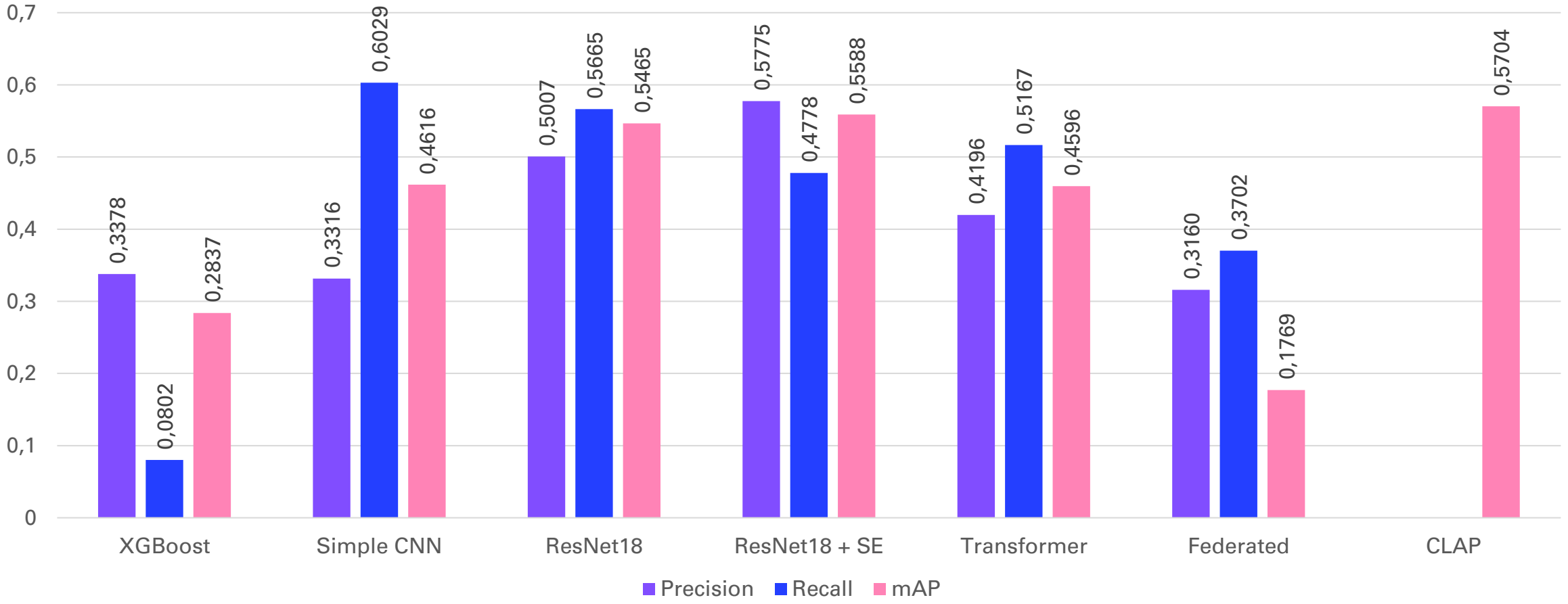


SUMMARY

Model Evaluation

Model	Best Precision	Best Recall	Best mAP
XGBoost	0.3378	0.0802	0.2837
Simple CNN	0.3316	0.6029	0.4616
ResNet18	0.5007	0.5665	0.5465
ResNet18 + SE	0.5775	0.4778	0.5588
Transformer	0.4196	0.5167	0.4596
ResNet18 + FL	0.3160	0.3702	0.1769
CLAP	-	-	0.5704

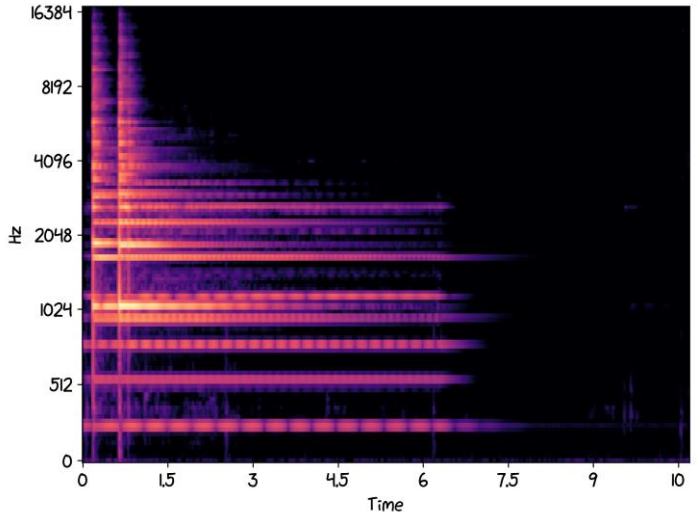
Model Evaluation



Label Comparison

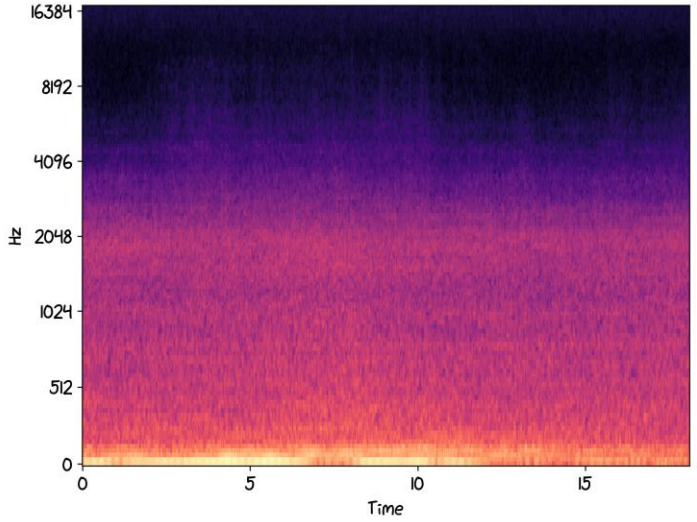
Good at predicting

- Applause
- Thunder
- Music
- Human Voice
- Toilet Flush
- Burping and Eruction



Bad at predicting

- Truck
- Bus
- Tick
- Screech
- Thump
- Thud



CONCLUSION AND OUTLOOK

APPENDIX

PROJECT GROUP

Bastian Holten-Møller – zxm415

Frida Ebdrup Jørgensen – grz328

Julie Rudkjær Walas – jqz760

Thea Borup Guntofte – xlr668

All participants contributed equally, and we exchanged ideas with the other group working with this dataset as well.

DATA

The data consists of sound files taken from the open data source FSD50K – link can be found below. The file format is WAV, which is a lossless waveform audio file format, that stores the raw sound without compression.

FSD50K Dataset

The data itself consists of 51.197 sound clips, which is unequally categorized into one or more of 200 subclasses. These can be divided into 6 higher level classes provided by Audioset Ontology. The data is split in two – developing (40.966 files) and evaluation (10.231 files). The developing set is further split into training (36796 files) and validation (4170 files). We did not use the latter split and just trained on all the developing files and used the evaluation files for evaluation of accuracy and such.

PREPROCESSING

As described on the previous slide, the data used is wav files, which we chose to convert to spectrograms, that is the input to the CNN – the normalisation of these and the actual spectrograms are explained in further detail on the next slide.

All the data was ~25 GB and to avoid taking up all the space in RAM and being able to utilize the faster calculations by parallelising and utilising the GPUs we had on hand, the data was converted to a cache and saved as .pt files on disc. These could then be loaded into a PyTorch DataSet on demand. In general, PyTorch has a lot of valuable functions for audio loading and manipulation.

PREPROCESSING - NORMALISATION

Audio clips were resampled to 22,050 kHz, converted to mono, and fixed to a uniform length of 5.96 seconds using repeat-padding and center-trimming.

Each clip was transformed into a Log-Mel spectrogram (128 mel bins, FFT window 2048, hop length 512, 100-10,000 Hz) and converted to decibels with an 80 dB dynamic range. Generally, a spectrogram is a short-time Fourier transform (STFT) of the data in the wav file. A mel spectrogram is a spectrogram where the frequencies are converted to the mel scale – a scale where equal distance in pitch sounds equally distant for the listener.

Spectrograms were then normalised using the training set mean ($\mu \sim 0$ dB) and standard deviation ($\sigma \sim 20$ dB), computed in a separate pass over all training samples.

PREPROCESSING - AUGMENTATION

To reduce overfitting on FSD50K, we applied a combination of waveform-level and spectrogram-level augmentations during training. At the waveform level, random gain scaling simulates variation in recording volume, and pitch shifting displaces all frequencies slightly up or down.

At the spectrogram-level we applied SpecAugment, which randomly masks parts of the spectrogram, which should force the model to recognise sounds even when a part of the structure is obscured.

We also tried "Mixup", which blends clips with some weights and applies the same weight to the combined label vector.

All augmentations were applied on-the-fly in the dataset's `__getitem__` method via a `transform` argument, so the on-disk spectrogram cache remained free from augmentations.

XGBOOST – BACK TO BASICS

MODEL: Librosa has feature extraction which allows us to reduce the dimensionality to 36 features (mean & std of the numerical features avaliable), which we were able to feed a boosted decision tree (BDT from XGBoost).

Through a combined analysis (SHAP, gain and permutation) the 15 most important features were determined, and the BDT was then trained on these. The results of this – the mAP score – can be seen on slide 10.

Precision: 0.3378, Recall: 0.0802. The low recall is expected. If these values are weighted by label frequency they are both remarkably higher suggesting that rare labels drag down the performance.

This model has random top performing labels (Applause, Musical instrument, music, thunderstorm, thunder, human voice). The CNN models is performing best in music.

LIMITATIONS:

- The complexity of the data is removed making it a simpler problem but also less precise.

CONCLUSION: The model is superficial, but the feature importance analysis could be used for optimization of a more complex model.

SIMPLE CNN

Model: SimpleCNN is what it sounds like. A simple first go at a CNN. It consists of three Conv2d with 3x3 Kernels, and with Batch Normalisation and ReLU between them. The first two blocks are followed by 2×2 max-pooling to down-sample the dimensions, while the channel depth grows from 1 to 32, 64, and finally 128. A Global Average Pool then collapses the spatial dimensions into a single 2048-dimensional vector, which is passed through Dropout and a final linear layer producing one logit per class.

Limitations:

It is a somewhat shallow model, and it is not particularly refined. It is also very prone to overfitting, even with data augmentation.

Conclusion:

Although it was a simple first go, it does not have too terrible accuracy.

RESNET18

Model: ResNet18 is a well-established architecture for image classification. We basically borrow the structure, but we wanted to compare it to our home-cooked SimpleCNN, so we trained the weights ourselves. Of course, you could also get the pretrained weights and use that as a starting point. However, there was some incompatibility with TorchVision because of our ROCm setup, so we were not able to try this.

Limitations: Without pretrained weights, ResNet18 needs to learn all its feature representations from FSD50K alone, which limits how much it can benefit from its extra depth compared to a pretrained version. It is also considerably heavier than SimpleCNN at around 11M parameters.

Conclusion: Despite training from scratch, ResNet18 outperformed SimpleCNN, suggesting that the deeper network genuinely did help even on audio data.

SQUEEZE AND EXCITATION

Model: SE-ResNet18 is a direct extension of ResNet18, with one addition inside each BasicBlock: a Squeeze-and-Excitation (SE) module. The SE module works in three steps: First it *squeezes* the spatial dimensions, producing a single value per channel that summarises how active that channel is overall. It then passes these through two small linear layers (*excitation*) that learn which channels are important for the current input. Finally, it scales each channel of the feature map by its learned weight, acting like a sort of attention.

Limitations: Basically, the same as the pure ResNet18. Could benefit from pretrained weights.

Conclusion: The SE blocks gave a nice little improvement over the plain ResNet18, for very few added parameters. So an almost free improvement!

TRANSFORMER

Model: The audio spectrogram transformer (AST) treats the spectrogram as a sequence rather than an image. The spectrogram is divided into non-overlapping 16×16 patches using a Conv2d. Each patch is projected to a 256-dimensional vector. A learnable CLS token is prepended to the sequence, and learned positional embeddings are added so each patch "knows" where it came from.

Limitations: Transformers need more training, so and are slower to train, so this was never able to beat the other models by being trained from scratch. It also needs more hyperparameter tuning, which we just did not have time for.

Conclusion: Despite training from scratch, the AST did work reliably enough. It was not as good as the other models, but it was far from terrible. (At least it beat XGBoost!)

FEDERATED LEARNING

METHOD: Federated learning (FL) trains a model across multiple clients without raw data ever leaving them. Each client trains locally and only shares model updates, which are aggregated into a shared global model using weighted averages. In our setup, the FSD50K training set is partitioned across 20 simulated clients by index, each holding an equal share of audio clips. Each round, a random subset of 5 clients trains locally for one epoch using Adam with cosine annealing, and their updates are averaged back into a global ResNet18 model weighted by number of local samples. The results can be seen on slide 15.

LIMITATIONS:

- Index-based partitioning distributes data evenly but does not guarantee balanced class representation across clients.
- Federated training converges more slowly than centralised training.

FEDERATED LEARNING EXTRAS

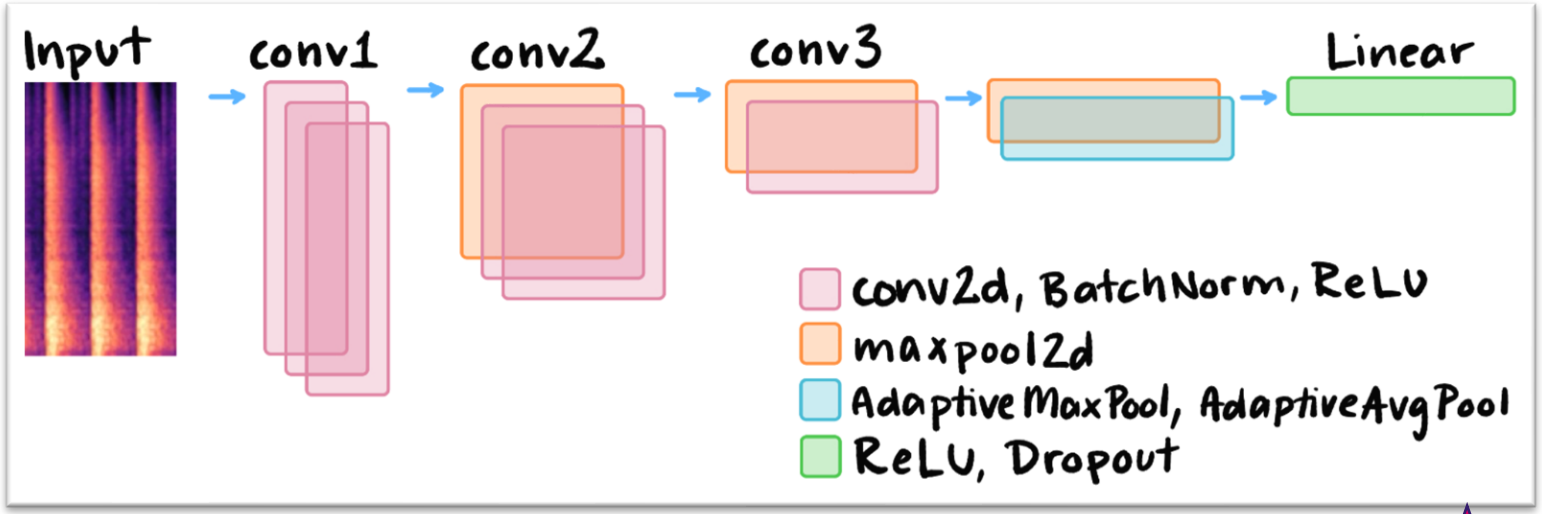
We also tried FL on two other models: Convolutional Recurrent NN and VGG which is a CNN made by the Visual Geometry Group from the University of Oxford which can be seen on the next slide.

However, none of these models were any good. The best model was CRNN yielding a mAP score of 0.0357 which reflects poor ranking ability. Some of the reasons this might not be working could be:

- *Instability in federated learning*: client models learn from different data distributions, and averaging their updates can degrade the global model.
- *CRNN model complexity*: the recurrent (GRU) part is sensitive to training conditions and can produce unstable or noisy outputs

Moreover, we did not have time to improve this further because we chose to focus on other models. To improve the model, we could have tuned the learning rate better and looked closer at the federated average.

OTHER MODELS USED

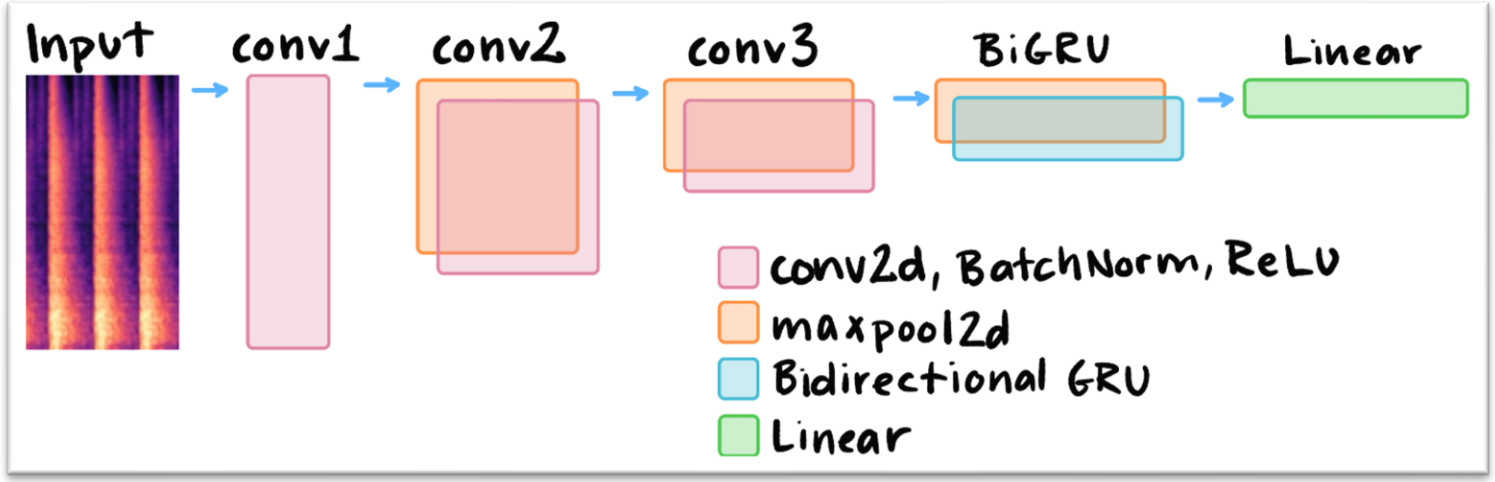


VGG (Visual Geometry Group Network)

- CNN architecture using stacked 3x3 convolutions
- Uses pooling to reduce dimensions and learn hierarchy
- Effective for image and audio classification tasks

CRNN (Convolutional recurrent neural network)

- Combines CNNs and RNNs
- CNN extracts spatial features from spectrograms
- RNN captures temporal patterns over time
- Well suited for audio and sequential data



Contrastive Language-Audio Pretraining

MODEL: A pretrained model specifically for audio-files and spectrograms, which we then finish training on our audio data. In the model text and audio are encoded separately and projected into a shared space. Contrastive loss is then applied to push matched pairs closer and unmatched further apart. It was trained on 48 kHz wav files, and we finalized the model by training on sample rates (SR) of 22.05 kHz, 44.1 kHz and 48 kHz to see if there was any difference (see table on slide 16).

LIMITATIONS:

- Training on our own data was a slow process.
- A lot of our data was removed as we cut it to max 7 sec for each sound clip
- No aggregation/augmentation of data, just the original data

CONCLUSION: The model has the best mAP score for 44.1 kHz SR which contrasts with the pretrained data that had 48 kHz, but there are only small variations in the different SR. The general mAP scores are 0.57.

GPU ACCESS

Training was carried out in part on a Linux machine with a slightly outdated AMD GPU (6700xt). This forced us to use PyTorch 2.11.0 with support for ROCm 7.2, which is AMD's open-source alternative to NVIDIA's CUDA. While ROCm works, it lags behind CUDA in many ways and was a bit of a hassle to get working. Also, there was an issue with crashing, likely because of memory spikes between epochs.

Some group members had newer MacBooks with Apple Silicon GPUs, which we discovered PyTorch could utilize through MPS (Metal Performance Shaders). MPS was more straightforward to set up than ROCm, but not nearly as fast in some cases.

Despite the challenges to get the code running on GPUs, the speedup was well worth it. It was also a good challenge to make it run faster, since we had to properly employ data loaders.

LINK TO GITHUB