Using open source tools to analyse and recognise sounds

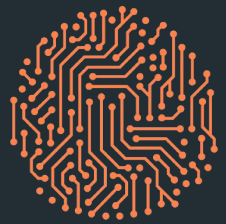
Theodoros Giannakopoulos

National Center for Scientific Research Demokritos

Institute of Informatics and Telecommunications - Computational Intelligence Lab

Multimodal Analysis Group (MagCIL)

labs-repos.iit.demokritos.gr/MagCIL



MagCIL

Multimodal Machine Learning

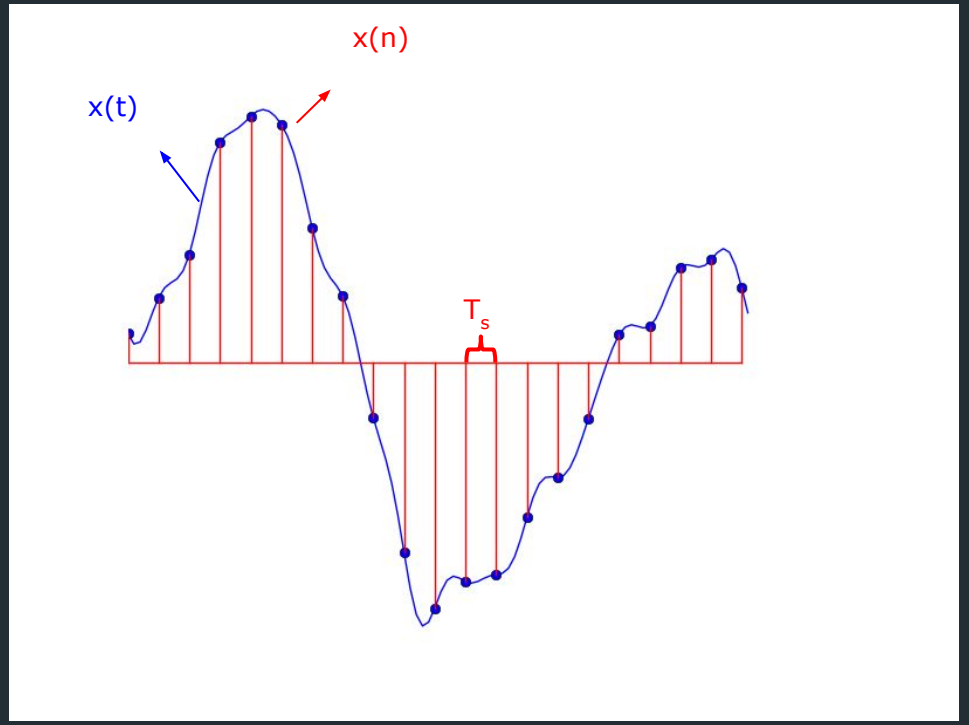
Presentation Objectives

- What is sound and how is it represented?
- How can we extract features from sounds?
- How one can build models that recognize between sound categories?
  - Using traditional ML techniques
  - Using DL methods
- Audio Analysis applications?

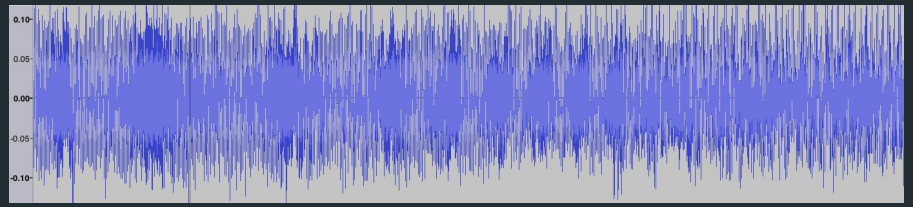
- Using open-source resources

What is sound?

- sound (physics):
  - a travelling vibration (wave)
  - through a medium (e.g. air)
  - transfers energy (particle to particle)
  - until “perceived” by our ears
- amplitude - loudness
- frequency - vibrations per sec
- analog sound → digital sound
  - sampling (sampling freq), fs
  - quantization (bits per sample)
  - Example:
    - 44100 Hz
    - 16 bits per sample (sample resolution)
    - ~8 million integers for an average song! (single channel….)

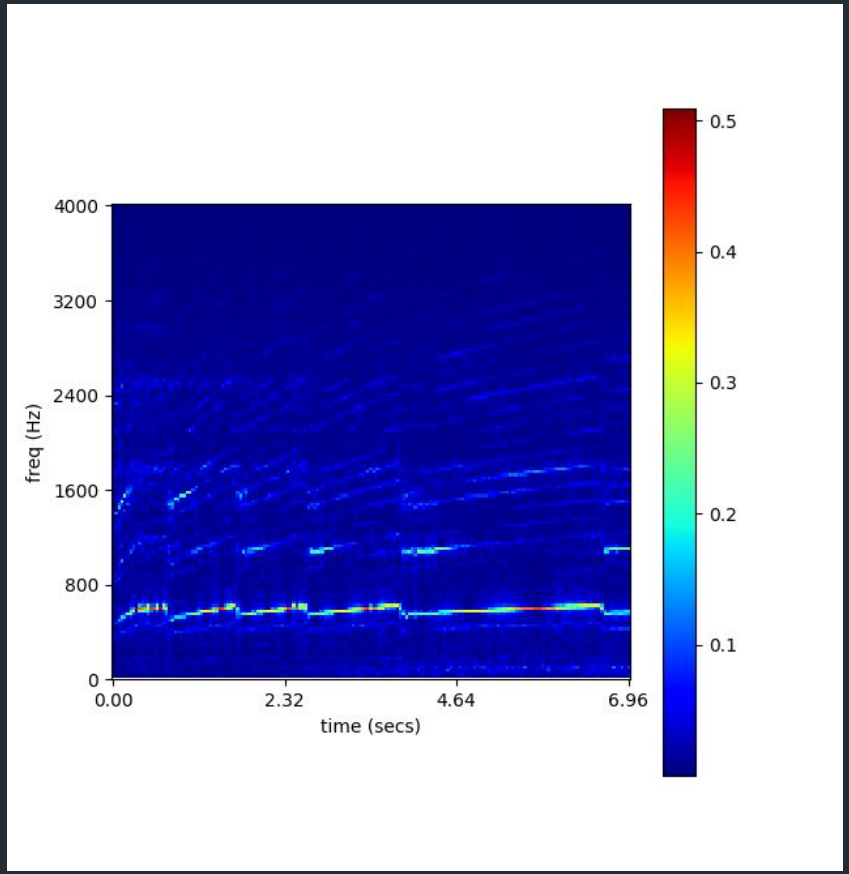


Frequency Representation

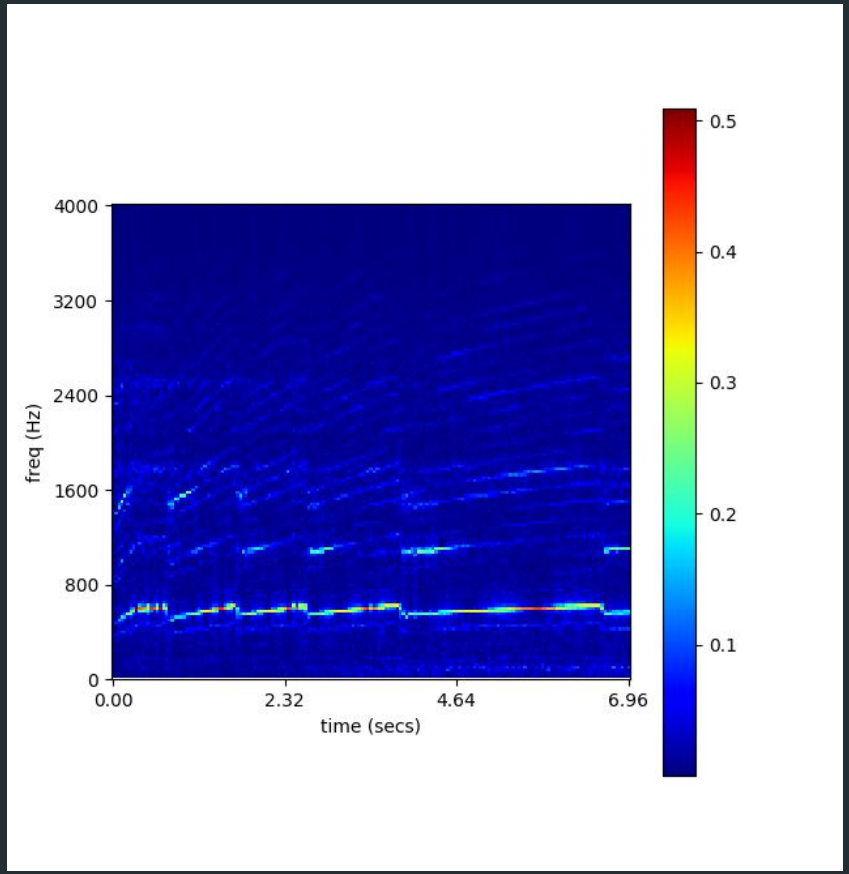


?

Frequency Representation



Frequency Representation



Python: load a sound, compute+plot its spectrogram

```
import numpy as np
import scipy.io.wavfile as wavfile
import librosa
import plotly.express as px
Fs, s = wavfile.read("sound2.wav")
s = np.double(s) / 2**15
S = np.abs(librosa.stft(s, int(Fs * 0.050), int(Fs * 0.050)))
fig = px.line(s)
fig.show()
fig = px.imshow(S[::-1, :])
fig.show()
```

Python: load a sound, compute+plot its spectrogram

```
import numpy as np
import scipy.io.wavfile as wavfile
import librosa
import plotly.express as px
Fs, s = wavfile.read("sound2.wav")
s = np.double(s) / 2**15
S = np.abs(librosa.stft(s, int(Fs * 0.050), int(Fs * 0.050)))
fig = px.line(s)
fig.show()
fig = px.imshow(S[::-1, :])
fig.show()
```

Scientific computation in Python

- N-dimensional arrays
- Numerical computing functions
- Linear Algebra
- ...

Python: load a sound, compute+plot its spectrogram

```
import numpy as np
import scipy.io.wavfile as wavfile
import librosa
import plotly.express as px
Fs, s = wavfile.read("sound2.wav")
s = np.double(s) / 2**15
S = np.abs(librosa.stft(s, int(Fs * 0.050), int(Fs * 0.050)))
fig = px.line(s)
fig.show()
fig = px.imshow(S[::-1, :])
fig.show()
```

Scientific computation in Python

- Numpy mostly for its types
- Scipy contains more numerical algorithms
- Here just used for IO but this is just one of the ways (after all WAV files have a simple format)
- ...

Python: load a sound, compute+plot its spectrogram

```
import numpy as np
import scipy.io.wavfile as wavfile
import librosa
import plotly.express as px
Fs, s = wavfile.read("sound2.wav")
s = np.double(s) / 2**15
S = np.abs(librosa.stft(s, int(Fs * 0.050), int(Fs * 0.050)))
fig = px.line(s)
fig.show()
fig = px.imshow(S[::-1, :])
fig.show()
```

Audio analysis in Python

- Audio feature extraction
- Basic filtering

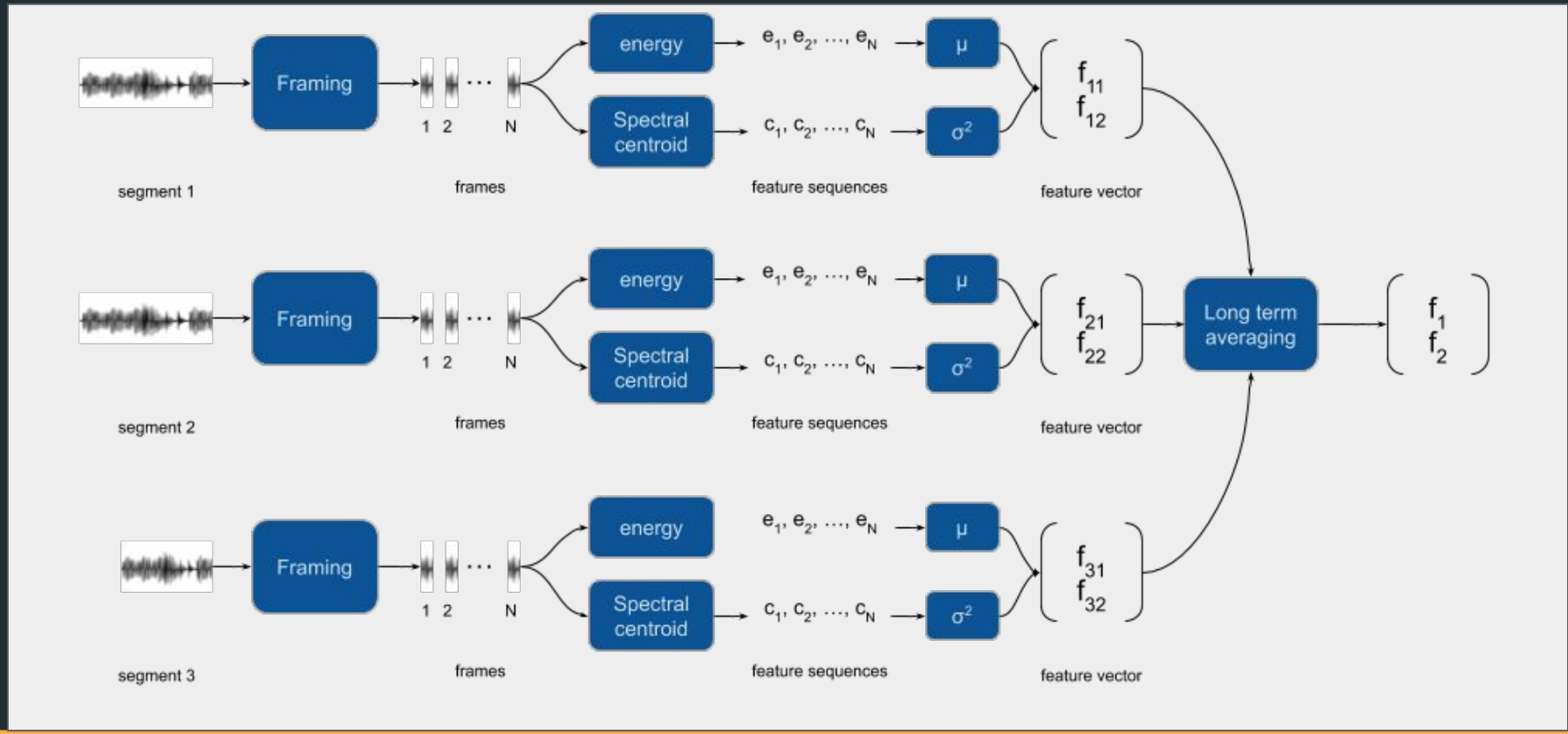
Python: load a sound, compute+plot its spectrogram

```
import numpy as np
import scipy.io.wavfile as wavfile
import librosa
import plotly.express as px
Fs, s = wavfile.read("sound2.wav")
s = np.double(s) / 2**15
S = np.abs(librosa.stft(s, int(Fs * 0.050), int(Fs * 0.050)))
fig = px.line(s)
fig.show()
fig = px.imshow(S[::-1, :])
fig.show()
```

Interactive plotting library for Python

- plotly.com includes dash and ML pipelines lately
- alternative matplotlib

Audio Feature Extraction and Traditional ML



Audio Feature Extraction and Traditional ML

- Audio features:
  - Short-term: extract 68 features per frame (e.g. 50msec)
  - Segment-term: extract 2 statistics (mean, std) per segment → 136 feature statistics
- Each feature represented by a 168-D vector

Audio Feature Extraction and Traditional ML

- Audio features:
  - Short-term: extract 68 features per frame (e.g. 50msec)
  - Segment-term: extract 2 statistics (mean, std) per segment → 136 feature statistics
- Each feature represented by a 168-D vector


```python
from pyAudioAnalysis import MidTermFeatures as aF
import os
import numpy as np
import plotly.graph_objs as go
import plotly
dirs = ["classical", "metal"]
class_names = [os.path.basename(d) for d in dirs]
m_win, m_step, s_win, s_step = 1, 1, 0.1, 0.05
# segment-level feature extraction:
features = []
for d in dirs: # get feature matrix for each directory (class)
    f, files, fn = aF.directory_feature_extraction(d, m_win, m_step,
                                                   s_win, s_step)
    features.append(f)

# select 2 features and create feature matrices for the two classes:
f1 = np.array([features[0][:, fn.index('spectral_centroid_mean')],
               features[0][:, fn.index('energy_entropy_mean')]])
f2 = np.array([features[1][:, fn.index('spectral_centroid_mean')],
               features[1][:, fn.index('energy_entropy_mean')]])
# plot 2D features
plots = [go.Scatter(x=f1[0, :],  y=f1[1, :],
                    name=class_names[0], mode='markers',
                    marker=dict(size=10,color='rgba(255, 182, 193, .9)')),
         go.Scatter(x=f2[0, :], y=f2[1, :],
                    name=class_names[1], mode='markers',
                    marker=dict(size=10,color='rgba(100, 100, 220, .9)'))]
plotly.offline.iplot(go.Figure(data=plots,
                               layout=go.Layout(xaxis=dict(title="spectral_centroid_mean"),
                                                yaxis=dict(title="energy_entropy_mean"))))
```

Audio Feature Extraction and Traditional ML

- Audio features:
  - Short-term: extract 68 features per frame (e.g. 50msec)
  - Segment-term: extract 2 statistics (mean, std) per segment → 136 feature statistics
- Each feature represented by a 136-D vector

pyAudio Analysis

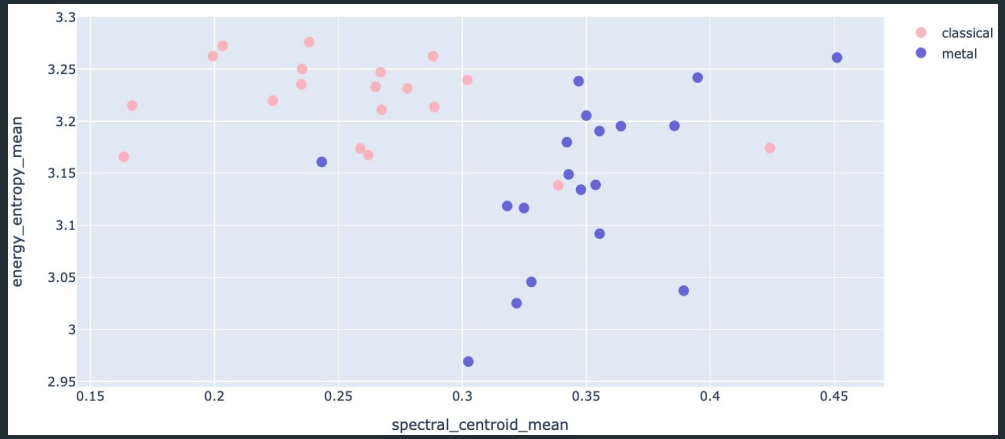
```
from pyAudioAnalysis import MidTermFeatures as aF
import os
import numpy as np
import plotly.graph_objs as go
import plotly
dirs = ["classical", "metal"]
class_names = [os.path.basename(d) for d in dirs]
m_win, m_step, s_win, s_step = 1, 1, 0.1, 0.05
# segment-level feature extraction:
features = []
for d in dirs: # get feature matrix for each directory (class)
    f, files, fn = aF.directory_feature_extraction(d, m_win, m_step,
                                                   s_win, s_step)
    features.append(f)

# select 2 features and create feature matrices for the two classes:
f1 = np.array([features[0][:, fn.index('spectral_centroid_mean')],
               features[0][:, fn.index('energy_entropy_mean')]])
f2 = np.array([features[1][:, fn.index('spectral_centroid_mean')],
               features[1][:, fn.index('energy_entropy_mean')]])
# plot 2D features
plots = [go.Scatter(x=f1[0, :],  y=f1[1, :],
                    name=class_names[0], mode='markers',
                    marker=dict(size=10,color='rgba(255, 182, 193, .9)')),
         go.Scatter(x=f2[0, :], y=f2[1, :],
                    name=class_names[1], mode='markers',
                    marker=dict(size=10,color='rgba(100, 100, 220, .9)'))]
plotly.offline.iplot(go.Figure(data=plots,
                               layout=go.Layout(xaxis=dict(title="spectral_centroid_mean"),
                                                yaxis=dict(title="energy_entropy_mean"))))
```

Audio Feature Extraction and Traditional ML

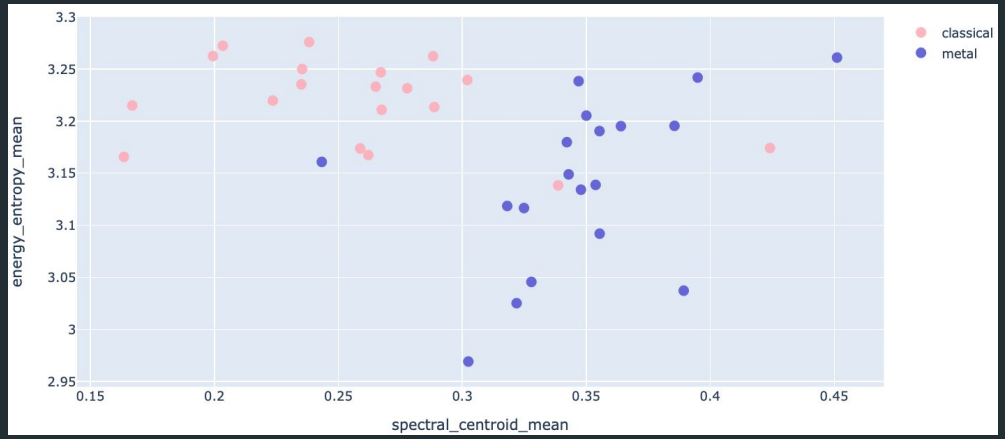
- Audio features:
  - Short-term: extract 68 features per frame (e.g. 50msec)
  - Segment-term: extract 2 statistics (mean, std) per segment → 136 feature statistics
- Each feature represented by a 136-D vector

pyAudio Analysis



Audio Feature Extraction and Traditional ML

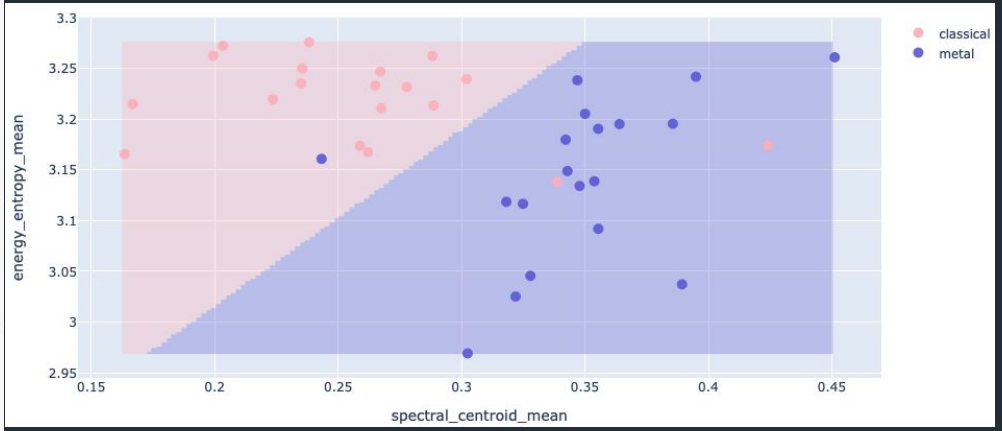
- Audio features:
  - Short-term: extract 68 features per frame (e.g. 50msec)
  - Segment-term: extract 2 statistics (mean, std) per segment → 136 feature statistics
- Each feature represented by a 136-D vector
- Then, you can use traditional ML - e.g. using scikit-learn



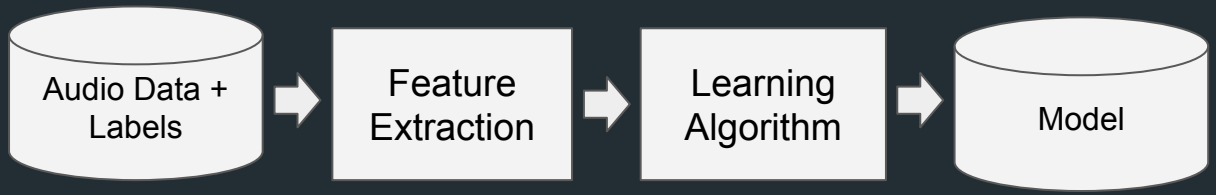
Audio Feature Extraction and Traditional ML

```python
from pyAudioAnalysis import MidTermFeatures as aF
import os
import numpy as np
from sklearn.svm import SVC
import plotly.graph_objs as go
import plotly
dirs = ["classical", "metal"]
class_names = [os.path.basename(d) for d in dirs]
m_win, m_step, s_win, s_step = 1, 1, 0.1, 0.05
# segment-level feature extraction:
features = []
for d in dirs: # get feature matrix for each directory (class)
    f, files, fn = aF.directory_feature_extraction(d, m_win, m_step,
                                                   s_win, s_step)
    features.append(f)

# select 2 features and create feature matrices for the two classes:
f1 = np.array([features[0][:, fn.index('spectral_centroid_mean')],
               features[0][:, fn.index('energy_entropy_mean')]])
f2 = np.array([features[1][:, fn.index('spectral_centroid_mean')],
               features[1][:, fn.index('energy_entropy_mean')]])
# plot 2D features
p1 = go.Scatter(x=f1[0, :],  y=f1[1, :], name=class_names[0],
                marker=dict(size=10,color='rgba(255, 182, 193, .9)'), mode='markers')
p2 = go.Scatter(x=f2[0, :], y=f2[1, :],  name=class_names[1],
                marker=dict(size=10,color='rgba(100, 100, 220, .9)'), mode='markers')
mylayout = go.Layout(xaxis=dict(title="spectral_centroid_mean"), yaxis=dict(title="energy_entropy_mean"))
y = np.concatenate((np.zeros(f1.shape[1]), np.ones(f2.shape[1])))
f = np.concatenate((f1.T, f2.T), axis = 0)
# train the svm classifier
cl = SVC(kernel='rbf', C=20)
cl.fit(f, y)
# apply the trained model on the points of a grid
x_ = np.arange(f[:, 0].min(), f[:, 0].max(), 0.002)
y_ = np.arange(f[:, 1].min(), f[:, 1].max(), 0.002)
xx, yy = np.meshgrid(x_, y_)
Z = cl.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape) / 2
# and visualize the grid on the same plot (decision surfaces)
cs = go.Heatmap(x=x_, y=y_, z=Z, showscale=False,
                colorscale= [[0, 'rgba(255, 182, 193, .3)'],
                             [1, 'rgba(100, 100, 220, .3)']])
mylayout = go.Layout(xaxis=dict(title="spectral_centroid_mean"),
                     yaxis=dict(title="energy_entropy_mean"))
plotly.offline.iplot(go.Figure(data=[p1, p2, cs], layout=mylayout))
```



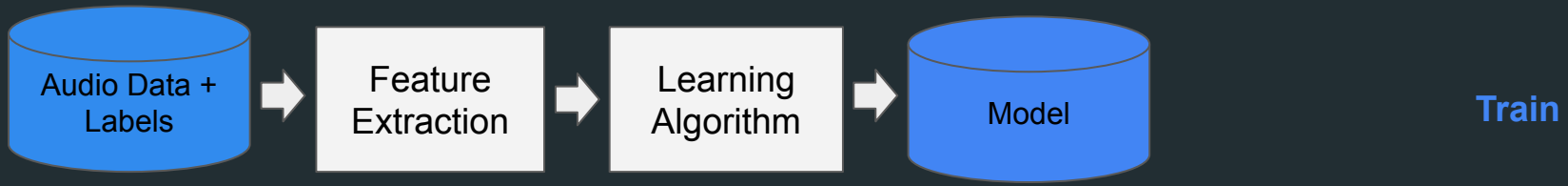
Train and test an audio classifier



Train

Train and test an audio classifier

```
from pyAudioAnalysis import audioTrainTest as at
at.extract_features_and_train(["classical", "metal"], 1, 1, 0.1, 0.1, "svm", "classical_vs_metal_model")
```



Train and test an audio classifier

Download two songs as examples

```
youtube-dl -f 140 https://www.youtube.com/watch\?v\=A8MO7fkZc5o
ffmpeg -i Metallica\ -\ Sad\ But\ True\ \(Official\ Music\ Video\)-A8MO7fkZc5o.m4a -ar 8000 -ac 1 -ss 100 -t 5 song1.wav

youtube-dl -f 140 https://www.youtube.com/watch\?v\=_4IRMYuE1hI
ffmpeg -i Beethoven\'s\ 5th\ Symphony-_4IRMYuE1hI.m4a -ar 8000 -ac 1 -ss 100 -t 5 song2.wav
```

Train and test an audio classifier

Download two songs as examples

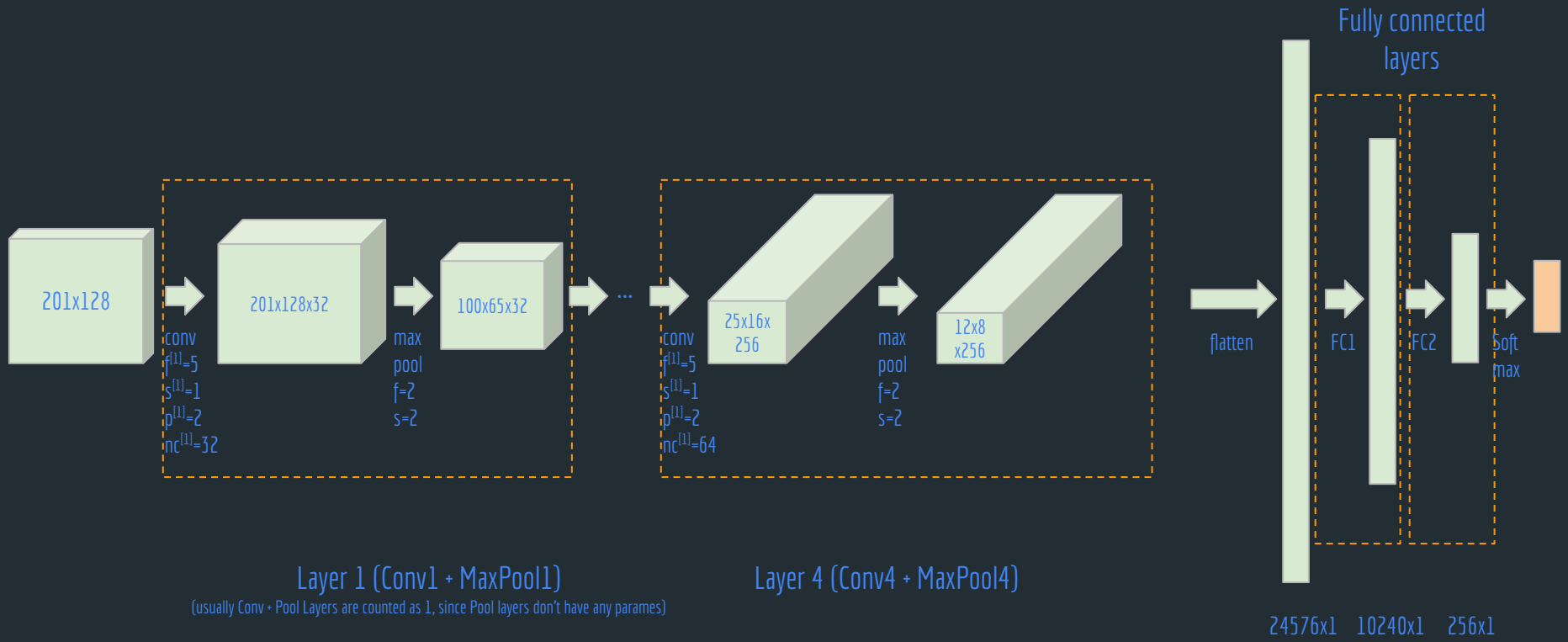
```
youtube-dl -f 140 https://www.youtube.com/watch\?v\=A8MO7fkZc5o
ffmpeg -i Metallica\ -\ Sad\ But\ True\ \(Official\ Music\ Video\)-A8MO7fkZc5o.m4a -ar 8000 -ac 1 -ss 100 -t 5 song1.wav

youtube-dl -f 140 https://www.youtube.com/watch\?v\=_4IRMYuE1hI
ffmpeg -i Beethoven\'s\ 5th\ Symphony-_4IRMYuE1hI.m4a -ar 8000 -ac 1 -ss 100 -t 5 song2.wav
```

```
In [9]: wclass, p, classes = at.file_classification("song1.wav", "classical_vs_metal_model", "svm"); print(classes[int(wclass)], p[int(wclass)])
metal 0.74058969762845

In [10]: wclass, p, classes = at.file_classification("song2.wav", "classical_vs_metal_model", "svm"); print(classes[int(wclass)], p[int(wclass)])
classical 0.6593364549202445
```

Deep Learning Example: Spectrograms + CNNs



Deep Learning Example: Spectrograms + CNNs

```
(conv_layer1): Sequential(
  (0): Conv2d(1, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): LeakyReLU(negative_slope=0.01)
  (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
)
(conv_layer2): Sequential(
  (0): Conv2d(32, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): LeakyReLU(negative_slope=0.01)
  (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
)
(conv_layer3): Sequential(
  (0): Conv2d(64, 128, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): LeakyReLU(negative_slope=0.01)
  (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
)
(conv_layer4): Sequential(
  (0): Conv2d(128, 256, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): LeakyReLU(negative_slope=0.01)
  (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
)
(linear_layer1): Sequential(
  (0): Dropout(p=0.75, inplace=False)
  (1): Linear(in_features=24576, out_features=1024, bias=True)
  (2): LeakyReLU(negative_slope=0.01)
)
(linear_layer2): Sequential(
  (0): Dropout(p=0.5, inplace=False)
  (1): Linear(in_features=1024, out_features=256, bias=True)
  (2): LeakyReLU(negative_slope=0.01)
)
(linear_layer3): Sequential(
  (0): Linear(in_features=256, out_features=16, bias=True)
  (1): LeakyReLU(negative_slope=0.01)
)
```

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 32, 201, 128]	832
BatchNorm2d-2	[-1, 32, 201, 128]	64
LeakyReLU-3	[-1, 32, 201, 128]	0
MaxPool2d-4	[-1, 32, 100, 64]	0
Conv2d-5	[-1, 64, 100, 64]	51,264
BatchNorm2d-6	[-1, 64, 100, 64]	128
LeakyReLU-7	[-1, 64, 100, 64]	0
MaxPool2d-8	[-1, 64, 50, 32]	0
Conv2d-9	[-1, 128, 50, 32]	204,928
BatchNorm2d-10	[-1, 128, 50, 32]	256
LeakyReLU-11	[-1, 128, 50, 32]	0
MaxPool2d-12	[-1, 128, 25, 16]	0
Conv2d-13	[-1, 256, 25, 16]	819,456
BatchNorm2d-14	[-1, 256, 25, 16]	512
LeakyReLU-15	[-1, 256, 25, 16]	0
MaxPool2d-16	[-1, 256, 12, 8]	0
Dropout-17	[-1, 24576]	0
Linear-18	[-1, 1024]	25,166,848
LeakyReLU-19	[-1, 1024]	0
Dropout-20	[-1, 1024]	0
Linear-21	[-1, 256]	262,400
LeakyReLU-22	[-1, 256]	0
Linear-23	[-1, 16]	4,112
LeakyReLU-24	[-1, 16]	0

Total params: 26,510,800

Deep Learning

- github.com/tyiannak/deep_audio_features
- Uses Pytorch for audio classifiers
- Train as simple as

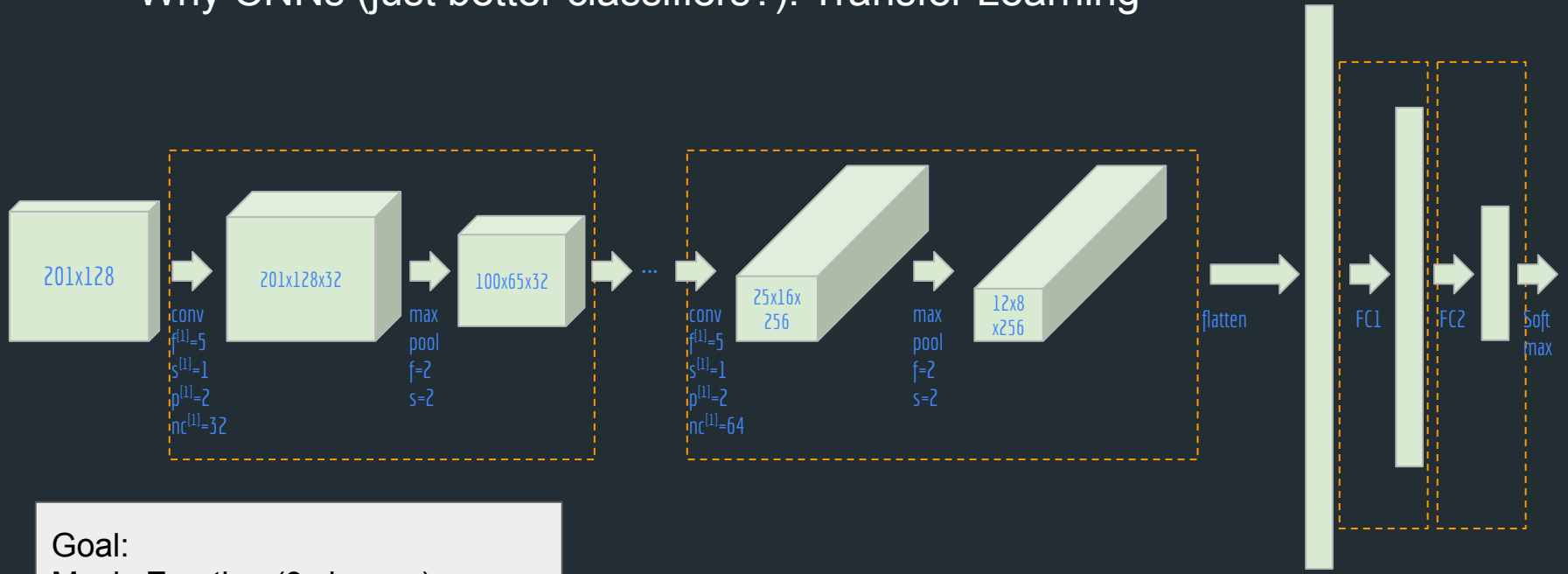
```
from deep_audio_features.bin import basic_training as bt

bt.train_model(["classical","metal"], "classical_vs_metal_cnn")
```

- Why CNNs (just better classifiers?)

Deep Learning

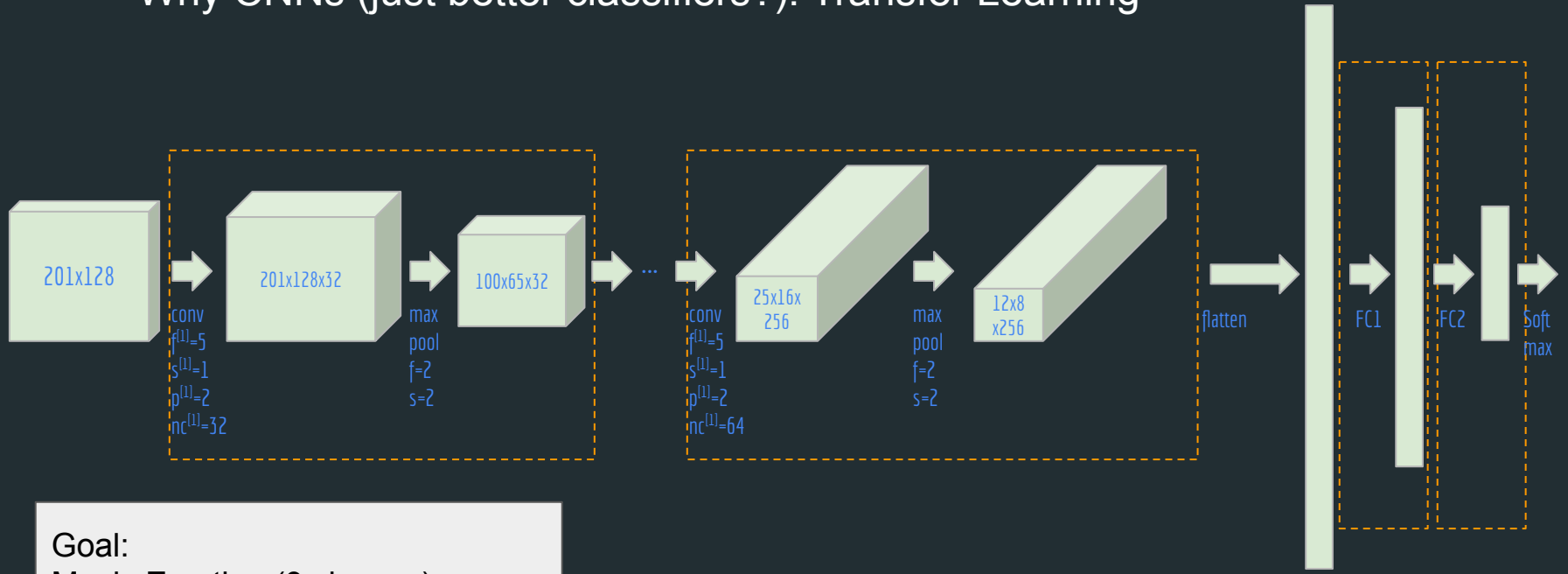
- Why CNNs (just better classifiers?): Transfer Learning



Goal:
Music Emotion (3 classes)
1K training samples

Deep Learning

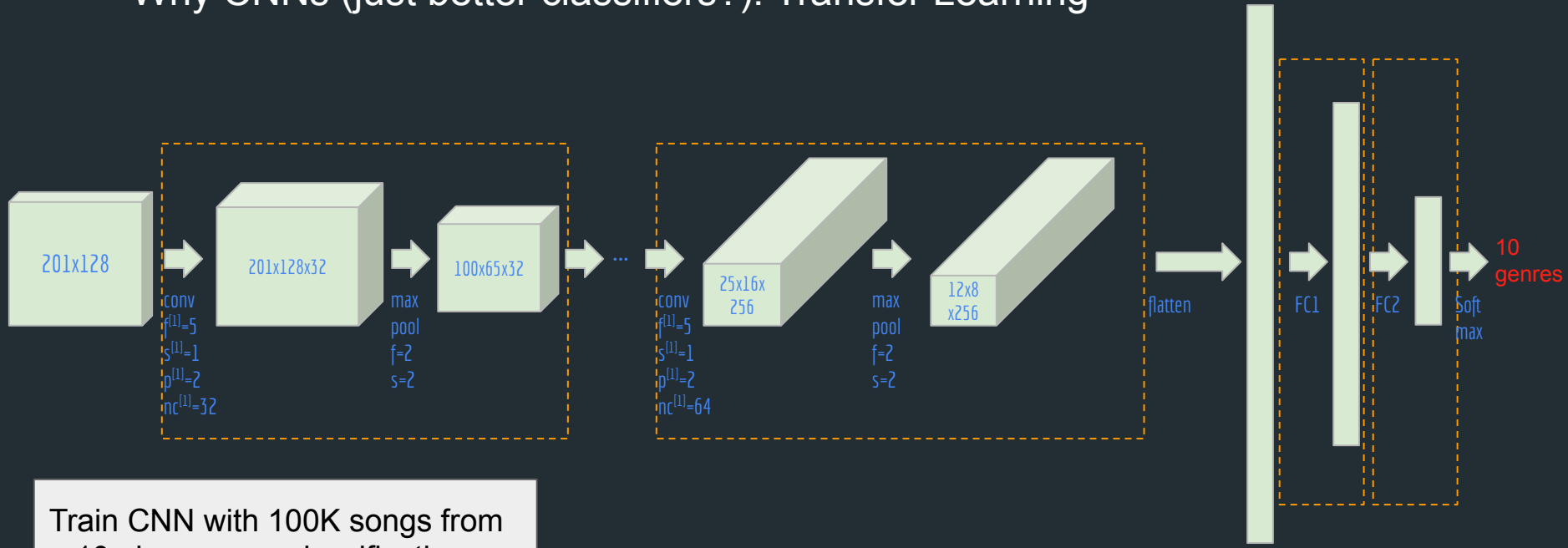
- Why CNNs (just better classifiers?): Transfer Learning



Goal:
Music Emotion (3 classes)
1K training samples → FEW

Deep Learning

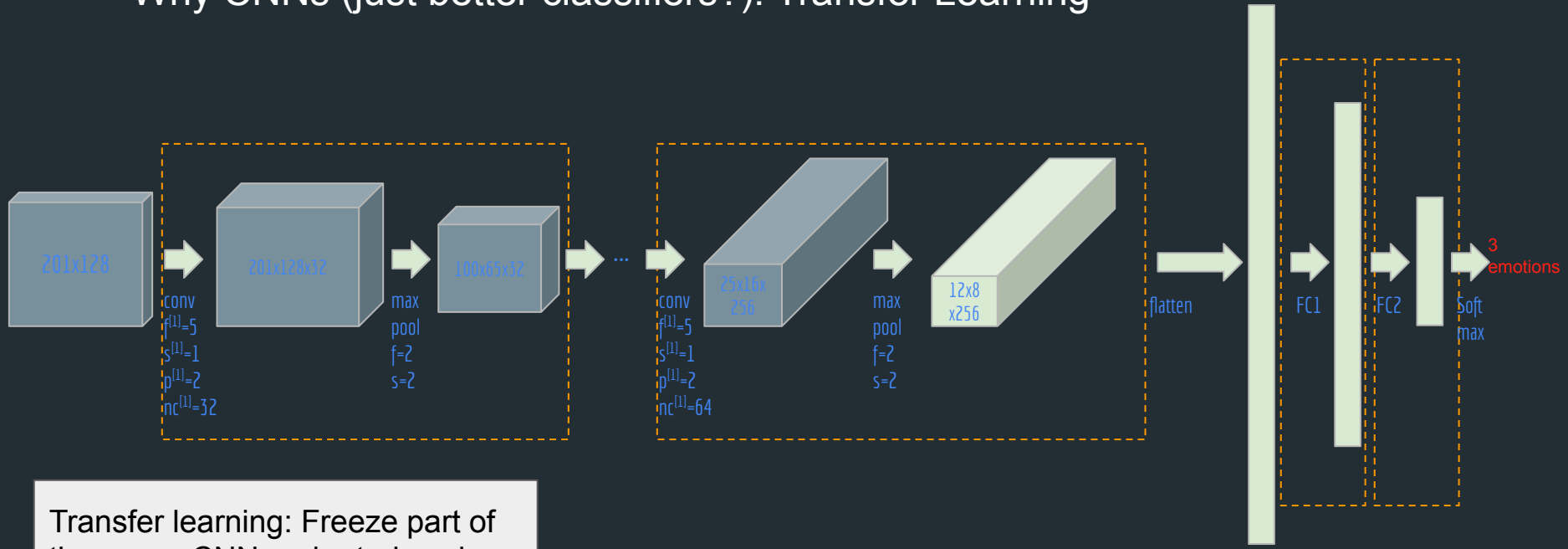
- Why CNNs (just better classifiers?): Transfer Learning



Train CNN with 100K songs from a 10-class genre classification task (easier access to data)

Deep Learning

- Why CNNs (just better classifiers?): Transfer Learning



Transfer learning: Freeze part of the genre-CNN and retrain using the 1K emotion data

Deep Learning

- Why CNNs (just better classifiers?): Transfer Learning

```
from deep_audio_features.bin import basic_training as bt

bt.train_model(["classical","metal", "jazz"], "classical_vs_metal_cnn")

from deep_audio_features.bin import transfer_learning as tl

tl.transfer_learning('classical_vs_metal_cnn.pt', ['negative', 'positive'] , strategy=0)
```

Applications

- Auditory event recognition for
  - security-surveillance
  - defence
  - environment
- Speaker recognition
- Speech emotion and behavior recognition
- Music information retrieval:
  - Mood and style analysis
  - Genre classification

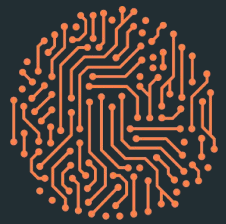
Resources

- pytorch.org/tutorials/beginner/audio_preprocessing_tutorial.html
- pytorch.org
- kaldi-asr.org
- github.com/tyiannak/pyAudioAnalysis
- github.com/tyiannak/deep_audio_features

Thank You

Questions?

magcil.github.io



MagCIL

Multimodal Machine Learning