

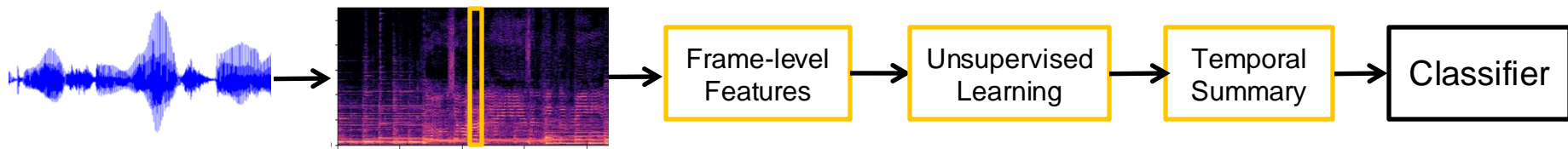
GCT634/AI613: Musical Applications of Machine Learning

Music Classification: Deep Learning



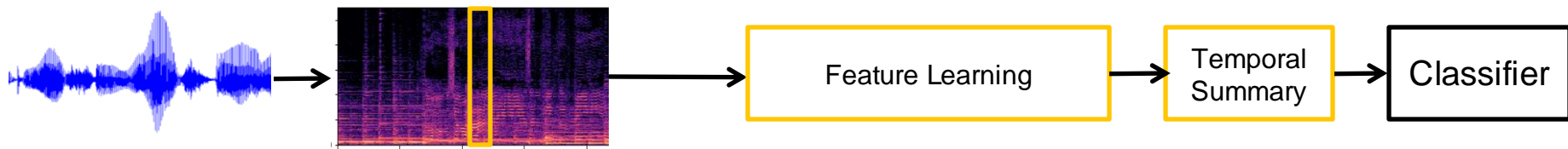
Juhan Nam

Traditional Music Classification



- Frame-level features are engineered based on domain knowledge and mainly capture short-term characteristics
- Each step of the pipeline is locally tuned
- “MFCC + K-means (VQ) + Codebook histogram” is still used as a baseline

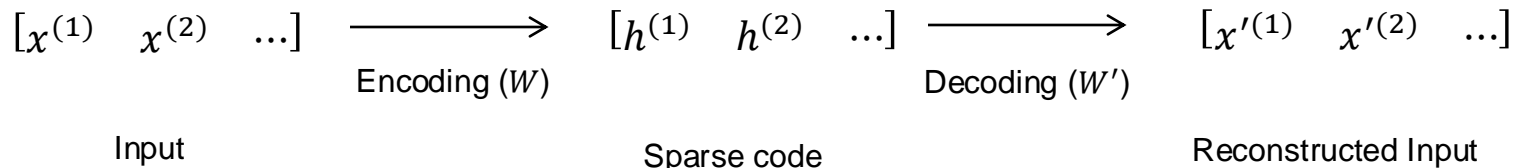
Early Deep Learning (2006 – 2012)



- From **feature engineering** to **feature learning**
- Learning audio features directly to the time-frequency representation
- Unsupervised learning: shallow and sparse
- Supervised learning: deep and dense

Early Deep Learning (2006 – 2012)

- Unsupervised learning: shallow and sparse feature learning
 - Learn a sparse feature representation h from the input x
 - The sparsity constraint on the encoded vector (h) encourages the meaningful pattern finding \rightarrow the input is explained by a few elements
 - The training (learning W) was often called “**dictionary learning**”
 - Examples of algorithms
 - K-means (h is one-hot vectors and W is cluster centers)
 - Sparse coding, sparse restricted Boltzmann machine (RBM), sparse auto-encoder



Early Deep Learning (2006 – 2012)

- Constant-Q transform (single frame) + Sparse Coding

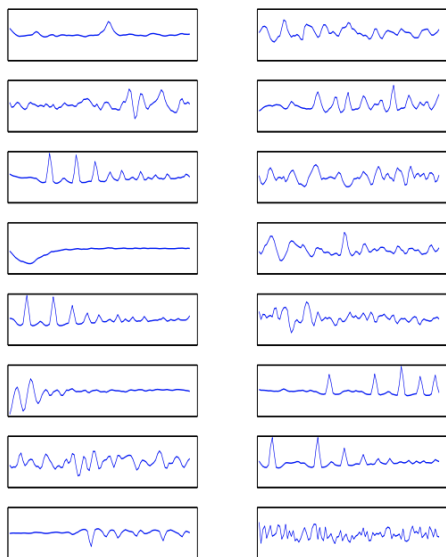


Figure 2. A random subset of the 512 basis functions learned on full CQT frames. The horizontal axis represents log-frequency and ranges from 67 Hz to 1046 Hz.

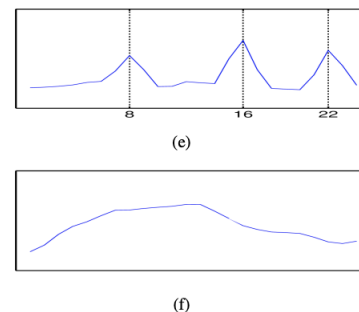
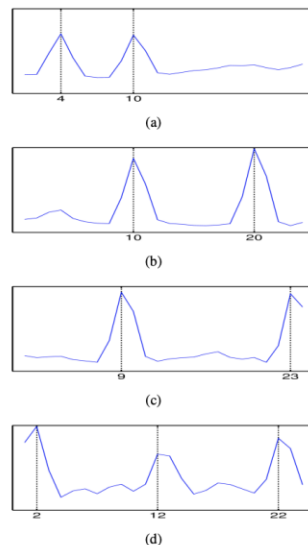


Figure 3. Some of the functions learned on individual octaves. The horizontal axis represents log-frequency. Recall that each octave consists of 24 channels a quarter tone apart. Channel numbers corresponding to peaks are indicated. a) A minor third (two notes 3 semitones apart) b) A perfect fourth (two notes 5 semitones apart) c) A perfect fifth (two notes 7 semitones apart) d) A quartal chord (each note is 5 semitones apart) e) A major triad f) A percussive sound.

Early Deep Learning (2006 – 2012)

- Mel-spectrogram (multi frames) + Sparse Coding/K-means/Sparse RBM

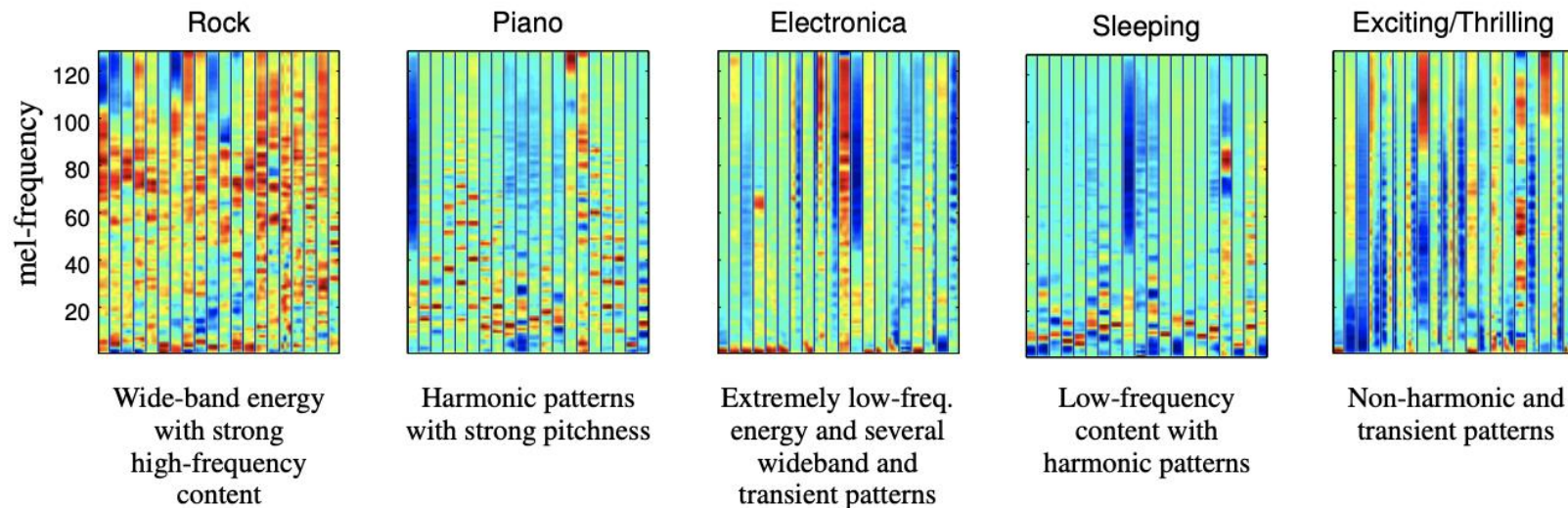
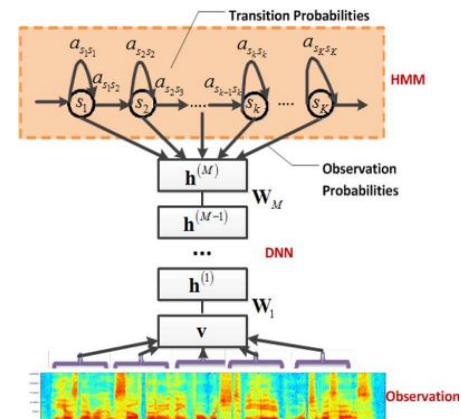


Figure 2: Top 20 most active feature bases (dictionary elements) for five different tags: *Rock*, *Piano*, *Electronica*, *Sleeping* and *Exciting/Thrilling*. Note that all the features come from the same learned dictionary (mel-frequency spectrogram and sparse RBM with 1024 hidden units and 0.01 sparsity), but different types of music use different feature bases.

Early Deep Learning (2006 – 2012)

- Supervised learning: deep and dense feature learning
 - Use a deep stacks of fully-connected layers (e.g.10 layers).
 - This “deep” neural network (DNN) was not trained well with random initialization
 - Instead, initialize the weights matrices using deep belief network (DBN)
 - Greedy layer-wise learning with a stack of RBMs
 - Then, supervised learning (fine-tuning the DNN) with labels
- The breakthrough in automatic speech recognition
 - The previous state-of-the-art model was based on GMM-HMM
 - The GMM module was replaced with the DNN-DBN



Early Deep Learning (2006 – 2012)

- Spectrogram (single frame)+ Deep Belief Network + Finetuning

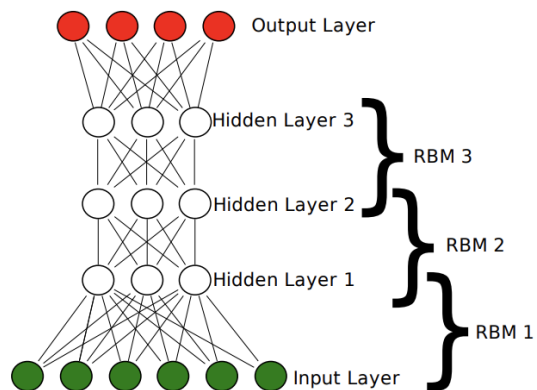
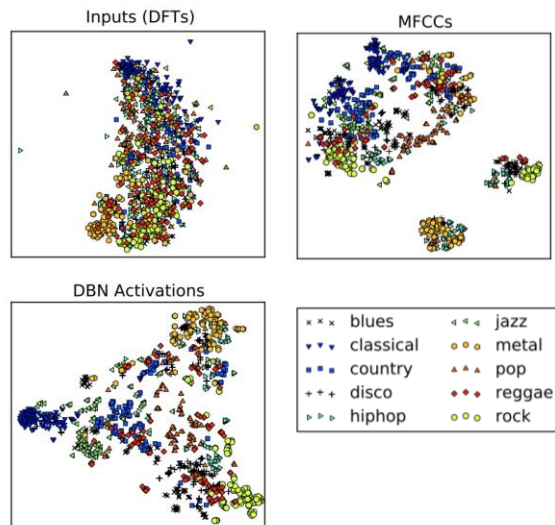


Figure 1. Schematic representation of a DBN. The number of layer and the number of units on each layer in the schema are only examples. We do not require to have the same number of units on each hidden layer.



	Accuracy
MFCCs	0.630
Layer 1	0.735
Layer 2	0.770
Layer 3	0.735
All Layers	0.770

Table 2. Classification accuracy for frame-level features

	Accuracy
MFCCs	0.790
Layer 1	0.800
Layer 2	0.837
Layer 3	0.830
All Layers	0.843

Table 3. Classification accuracy for features aggregated over 5 seconds

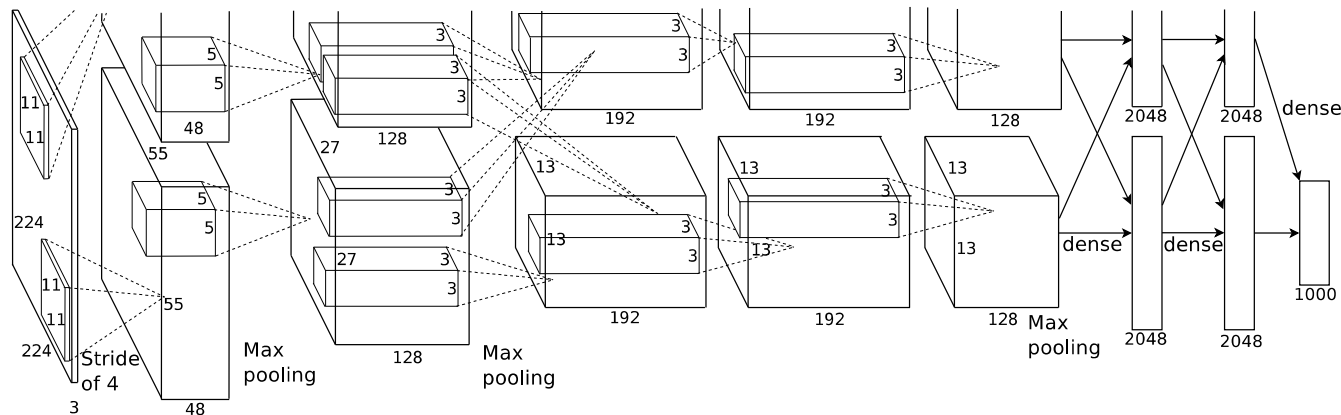
Convolutional Neural Network (2012 -)



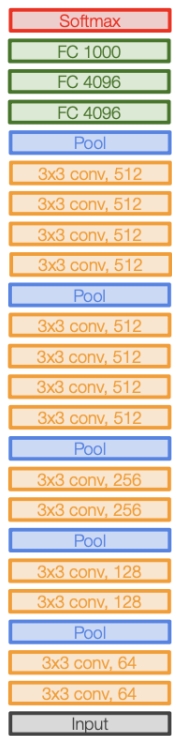
- End-to-end feature learning from the input data: error back-propagation
- Regard the time-frequency representation as a “2D image” and apply image classification models to it
- Supervised learning using labeled data

Convolutional Neural Network (2012 -)

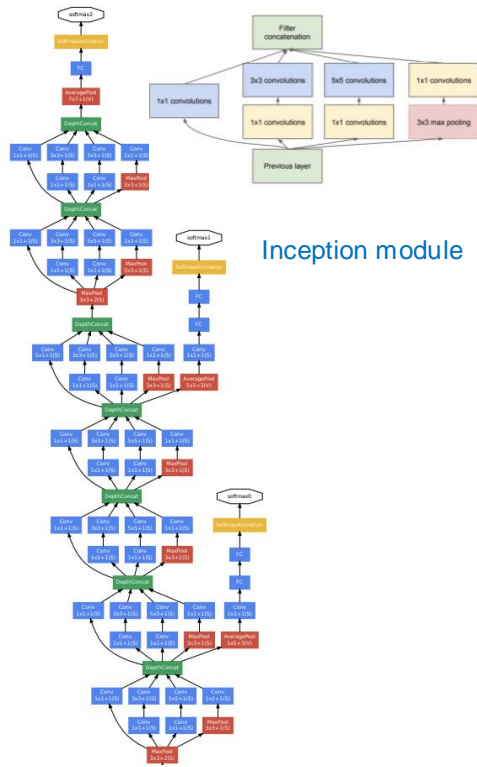
- AlexNet: the breakthrough in image classification (2012)
 - A stack of convolutional and max-pooling layers
 - ReLU (fast and non-saturated), dropout (regularization)
 - Trained with 2 GPUs on 1.2M images during one week
 - ImageNet challenge: top-5% error 15.3% (>10% lower than the second)



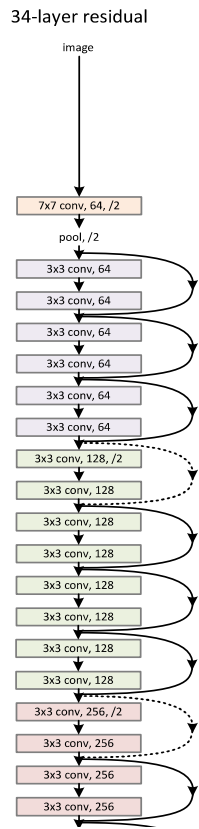
CNN Architectures for Image Classification



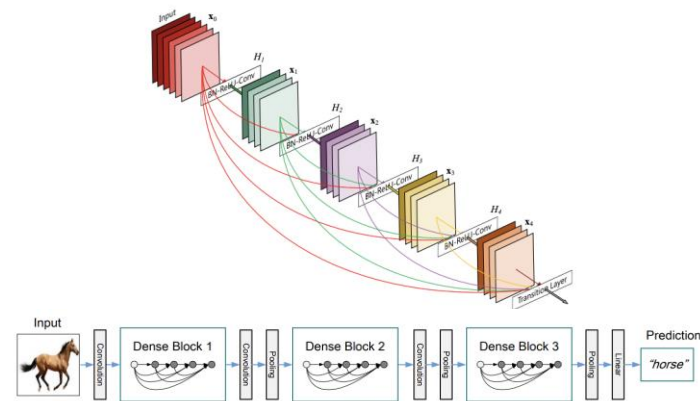
VGG



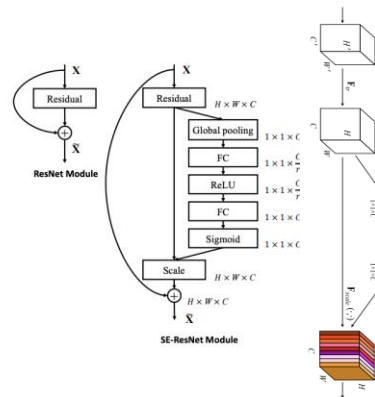
GoogleNet



ResNet



DenseNet

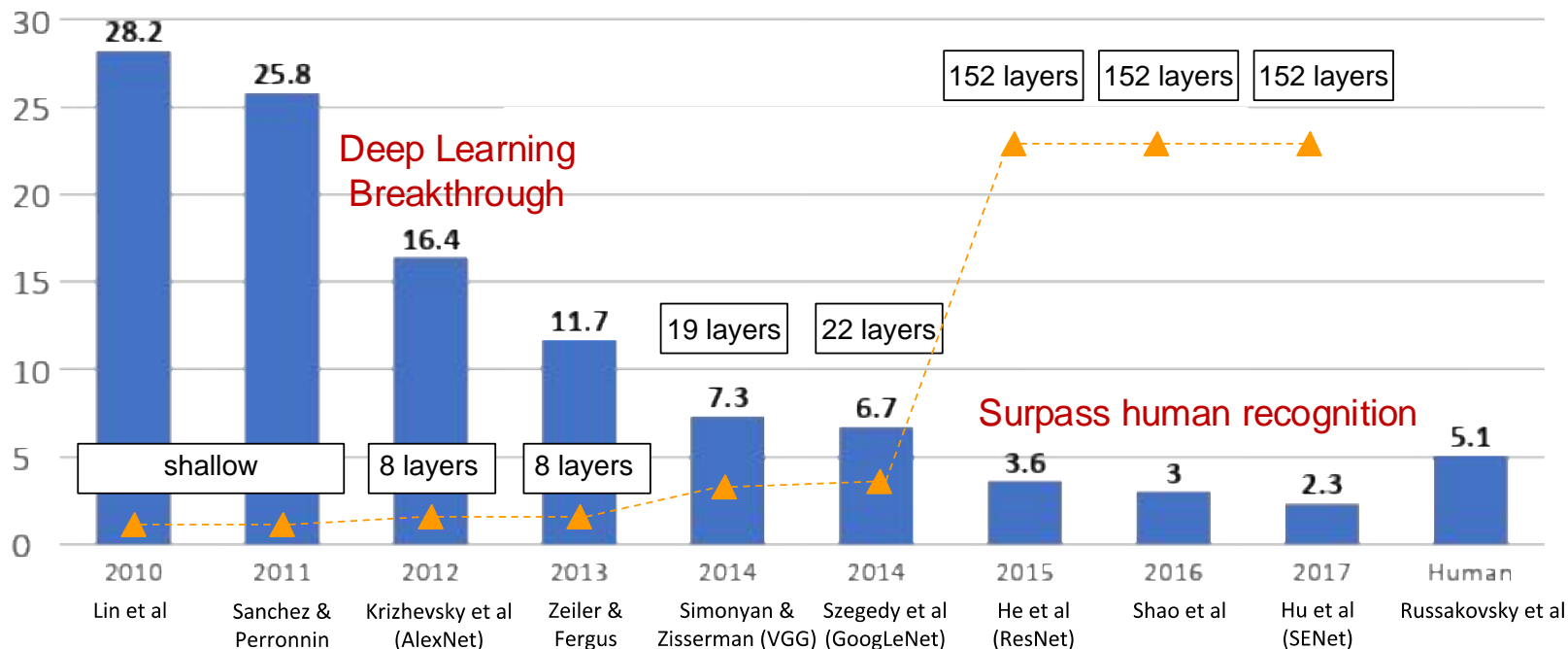


Squeeze-and-Excitation Net (SENet)

EfficientNet
MobileNet

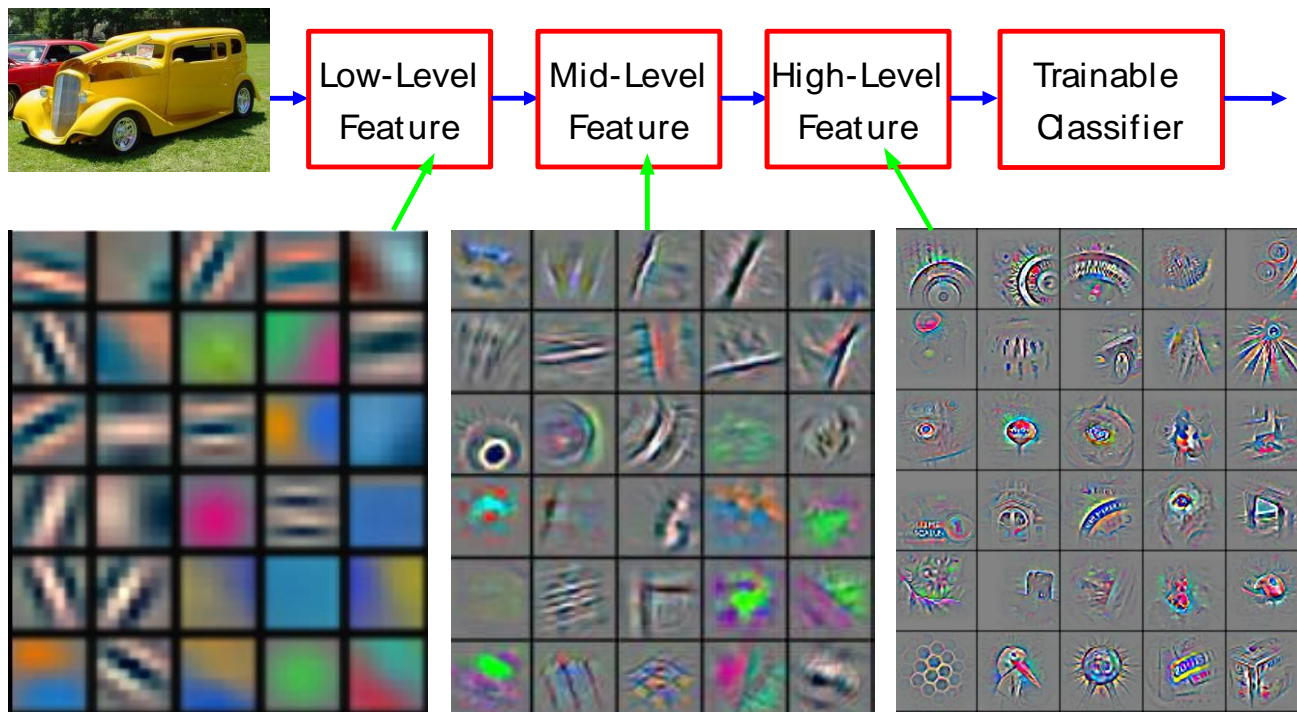
Convolutional Neural Network (2012 -)

- CNN models have been deeper and deeper



Hierarchical Representation Learning

- Learned features are similar to those in the human visual system



Deep Learning: Building Models

- **Parametric modules**

- Fully-connected
- Convolutional
- Skip / Residual
- Recurrent
- Attention
- Pooling

- **Nonlinearity functions (Non-parametric modules)**

- Sigmoid
- Tanh
- ReLU and variations



Deep Learning: Training Models

- **Loss Function**

- Cross entropy (logistic loss)
- Hinge loss
- Maximum likelihood
- L2 (root mean square) and L1
- Adversarial
- Variational

- **Optimizers**

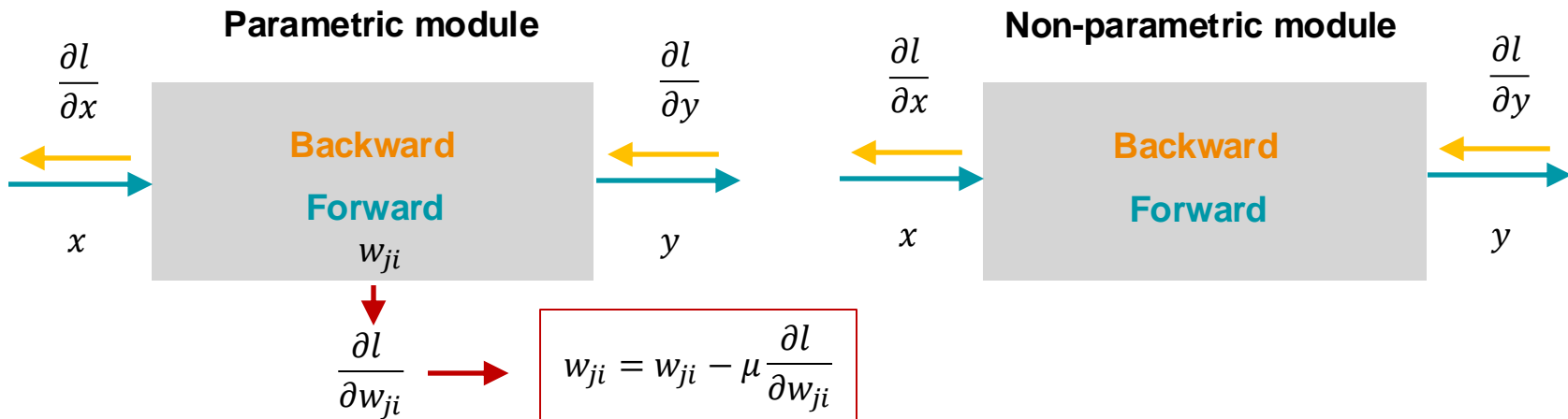
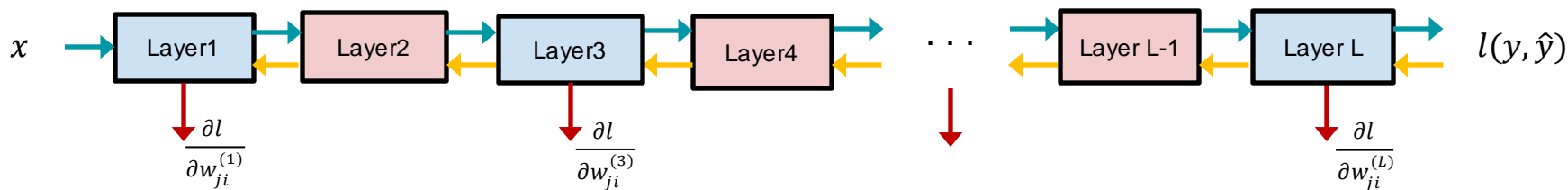
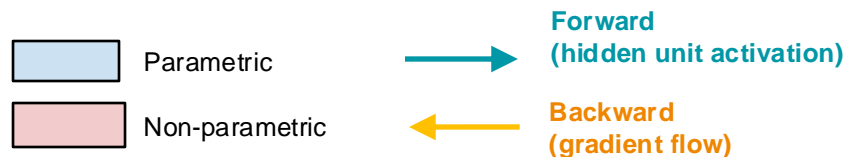
- SGD
- Momentum
- RMSProp
- Adagrad
- Adam

- **Hyper parameter**

- Weight initialization
- L1 and L2 (Weight decay)
- Dropout
- Learning rate
- Layer size
- Batch size
- Data augmentation

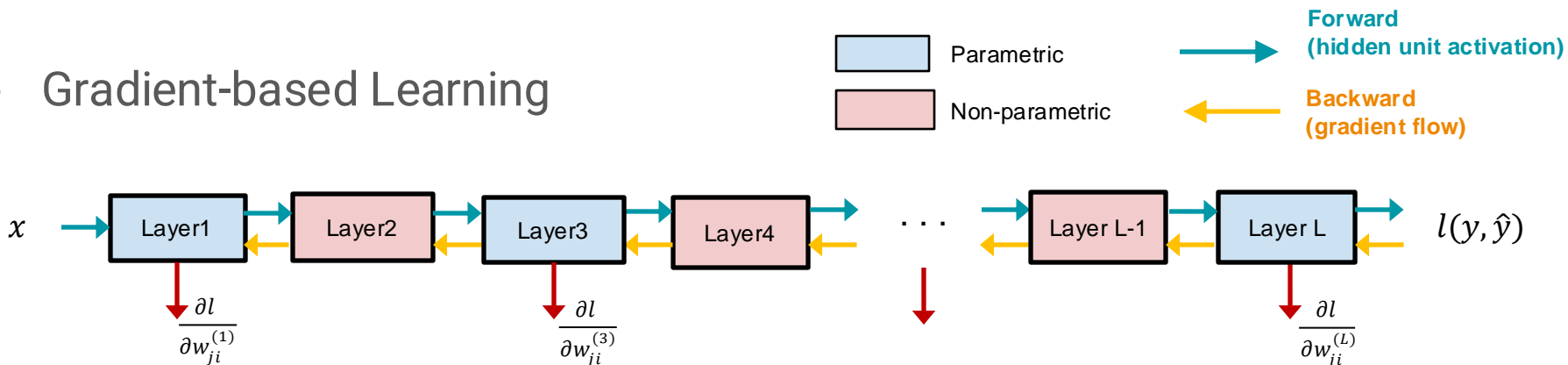
Training Deep Neural Networks

- Gradient-based Learning



Training Deep Neural Networks

- Gradient-based Learning



- **Vanishing gradient** or **exploding gradient** during the gradient flow
→ **the distribution of hidden unit activations in a controlled range**
 - **Input normalization**: zero mean and unit variance
 - **Weighted initialization**: to have the same variance between input and output at each layer to speed up the training (**Xavier or He initialization**)
 - **Batch normalization**: normalize the hidden units as a run-time processing during training

Optimization

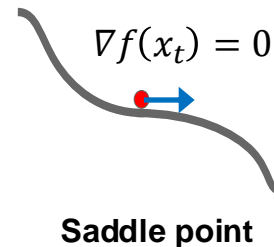
- Stochastic gradient descent is the basic optimizer in deep learning

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

- Nesterov Momentum

- Add the history of gradient to the parameter point

$$x_{t+1} = x_t + v_{t+1} \quad v_{t+1} = \rho v_t - \alpha \nabla f(x_t + \rho v_t)$$



- ADAM (**AD**aptive **M**omentum estimation)

- Use an adaptive learning rate for each parameter on top of the momentum
- Increase the learning rate for less updated parameters and vice versa

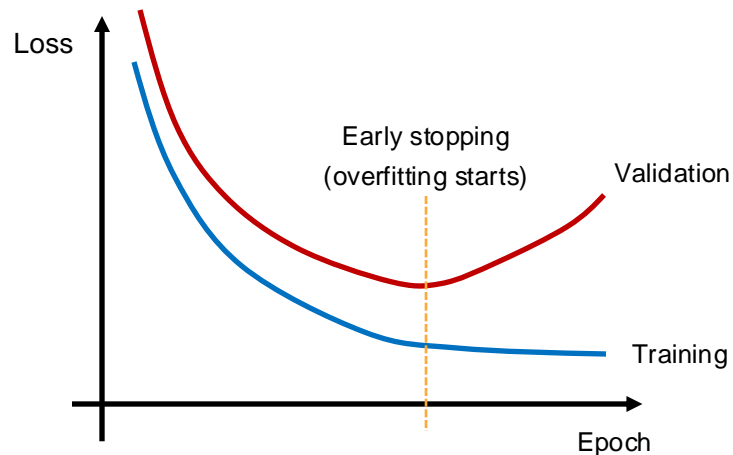
$$x_{t+1}(i) = x_t(i) - \frac{\alpha}{\sqrt{g_{t+1}(i) + \epsilon}} v_{t+1}$$

$$v_{t+1} = \rho v_t + (1 - \rho) \nabla f(x_t)$$

$$g_{t+1}(i) = \beta g_t(i) + (1 - \beta) \nabla f(x_t(i))^2$$

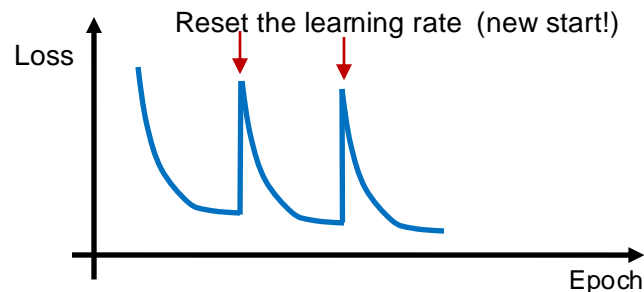
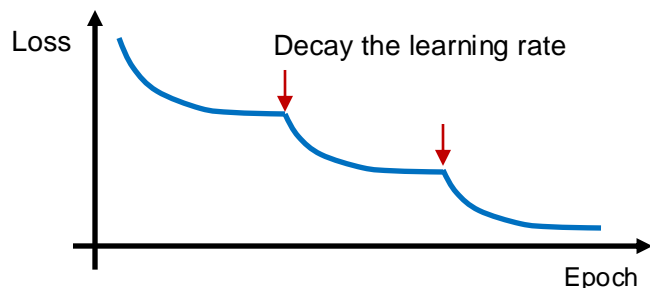
Training Deep Neural Networks

- Monitor both **training loss** and **validation loss** and stop the iteration when the validation loss decreases any more (**early stopping**)
 - Note that the training set is used for both the feedforward and backward passes whereas the validation set is used for only the feedforward pass



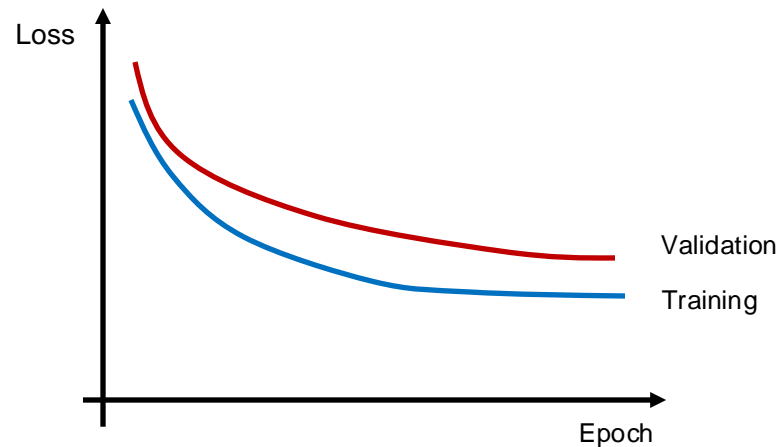
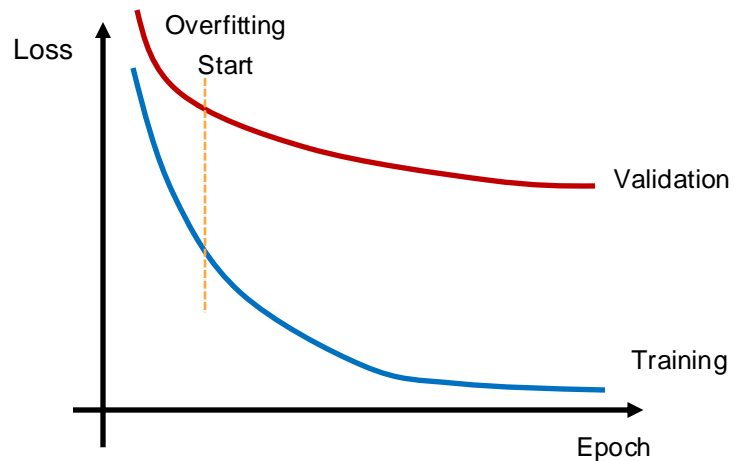
Annealing Learning Rate

- Decay the learning rate under certain conditions
 - Step decay: by a factor (e.g. 5 or 10) every fixed size of epoch
 - Exponential decay ($\alpha = \alpha_0 e^{-kt}$) or 1/t decay ($\alpha = \alpha_0 / (1+kt)$) is also possible
 - Reduce on plateau: decay around every early stopping point
- Reset the learning rate with “warm start”
 - Cosine (Loshchilov, 2017) and cyclic (Smith, 2017)
 - Start with a high learning rate and restart with better initial weights



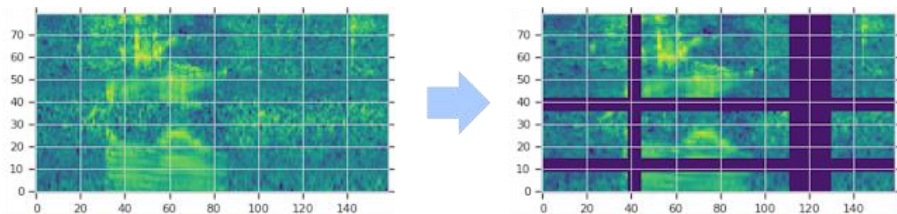
Regularization

- To avoid overfitting to the training data
 - Reducing the model size
 - Weight decay (L1, L2 norm of weights)
 - Dropout
 - Data augmentation



Data Augmentation

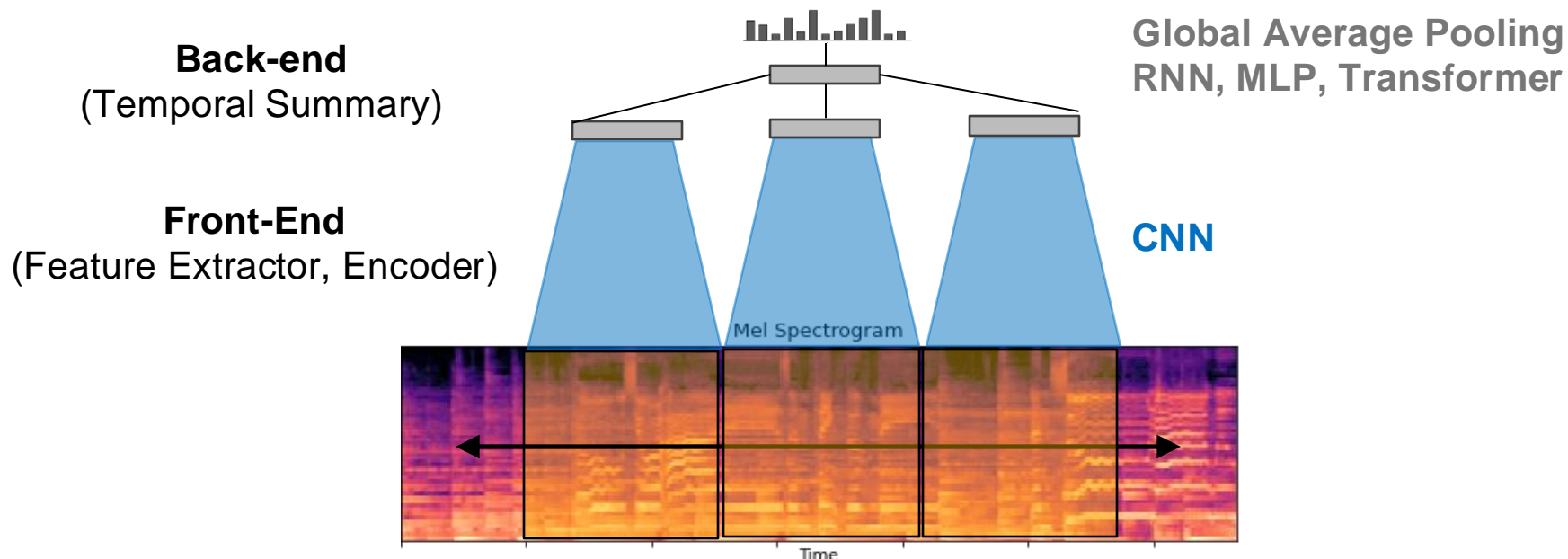
- A technique to increase the volume and variance of input data based on domain knowledge
 - Digital audio effects: pitch shifting, time-stretching, filters/EQ, delay, reverb
 - Adding noises: white/pink noise, babble/café noise
 - Random mask: SpecAugment



- Librosa, pysox, and torchaudio are commonly used
- Check if the output label is affected by the audio effects
 - You can also change the label according to the way of augmenting data


CNN for Music Classification

- Front-end: extract local audio features using CNN
- Back-end: summarize the local audio features over time



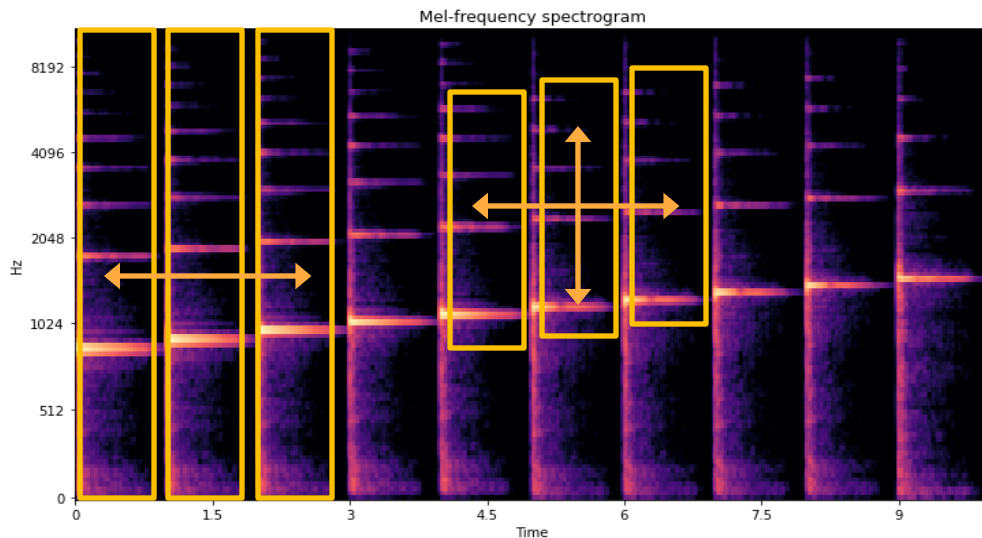
CNN for Music Classification

- Ground truth (or label) representation
 - Single-label classification: one-hot ($[0, 0, 1, 0, \dots, 0]$)
 - Multi-label classification: multi-hot ($[1, 0, 1, 1, \dots, 0]$)
- The output layer
 - Single-label classification: softmax
 - Multi-label classification: sigmoid
 - The loss function is defined as the cross-entropy between the sigmoid output ($\frac{1}{1+e^{-x}}$) and the multi-hot vector (e.g. $[1, 0, 1, 1, \dots, 0]$)

$$loss = - \sum_x p(x) \log q(x)$$


Convolutional Neural Network

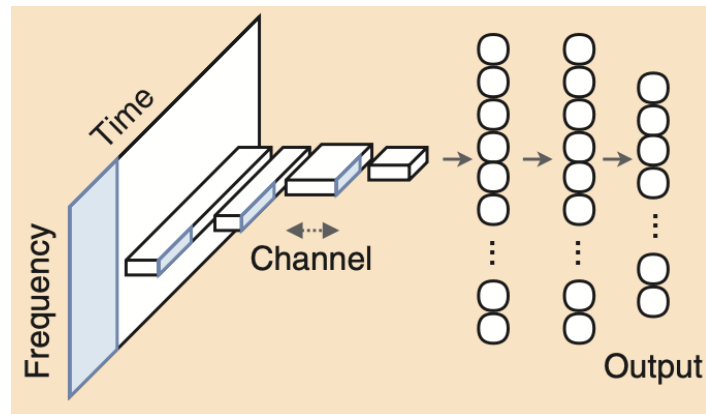
- The locality and translation invariance works in the audio domain
 - Both 1D and 2D translation are possible
 - 2D translation is only on the log-frequency scale which makes harmonic patterns approximately shifted-invariant



CNN for Music Classification: 1D CNN

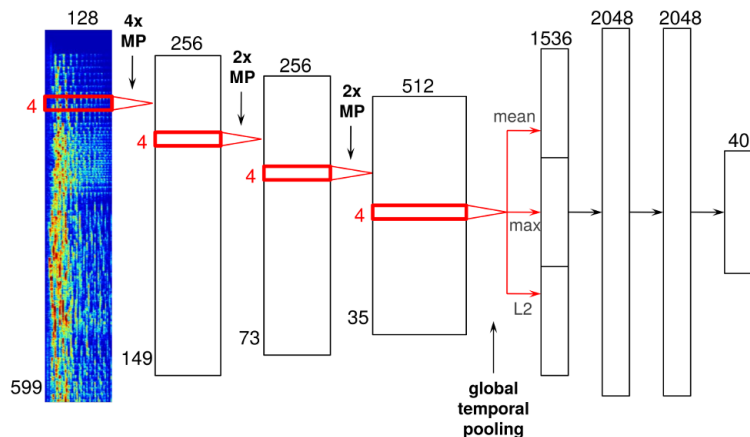
- 1D convolution blocks

- The filter size of the first conv layer covers the entire frequency range
- The 1D feature maps significantly reduce the number of parameters compared to the 2D feature map
- Fast to train
- Work well for small datasets
- Not invariant to pitch shifting:
key transpose changes the feature maps



CNN for Music Classification: 1D CNN

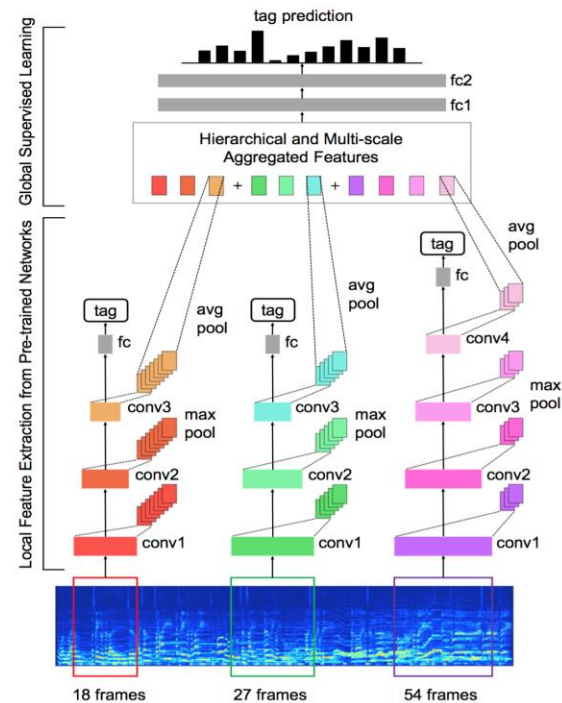
- Front-end: 1D CNN
 - Mel-spectrogram input filter with 128 (mel bin) x 4 (frames)
 - Feature maps (256 \rightarrow 256 \rightarrow 256 \rightarrow 512), max-pooling in time (4 \rightarrow 2 \rightarrow 2)
- Back-end: global pooling (temporal summary) with mean, max, and L2



<http://benanne.github.io/2014/08/05/spotify-cnns.html>

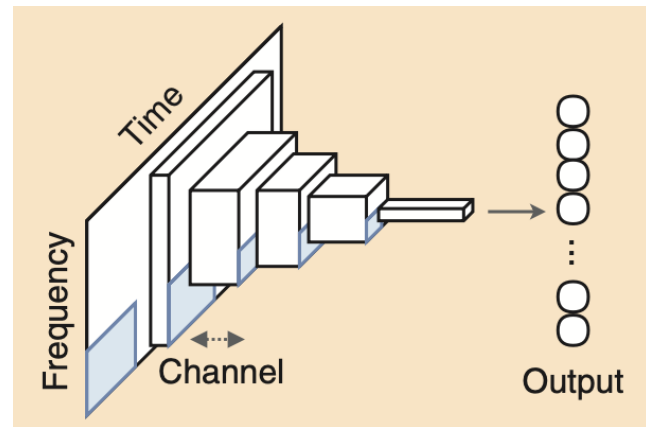
CNN for Music Classification: 1D CNN

- Front-end: multiple CNNs
 - Take different input sizes (small to wide contexts)
 - Trained independently
 - Each of them is 1D CNN
- Back-end: MLP
 - Take features from “all layers” of pre-trained CNNs and aggregate them with max/average pooling
 - MLP with 2 hidden layers makes a final prediction



CNN Architectures: 2D CNN

- 2D convolution blocks
 - Similar to CNN architectures for image classification: set the filter to be a time-frequency patch (e.g. 3x3) and it slides over both time and frequency
 - Relatively invariant to pitch shifting assuming that the input is a log-scaled spectrogram
 - 2D feature maps significantly increases the number of parameters and thus need more computational resources
 - But, it is more flexible and powerful
 - More commonly used than 1D CNN



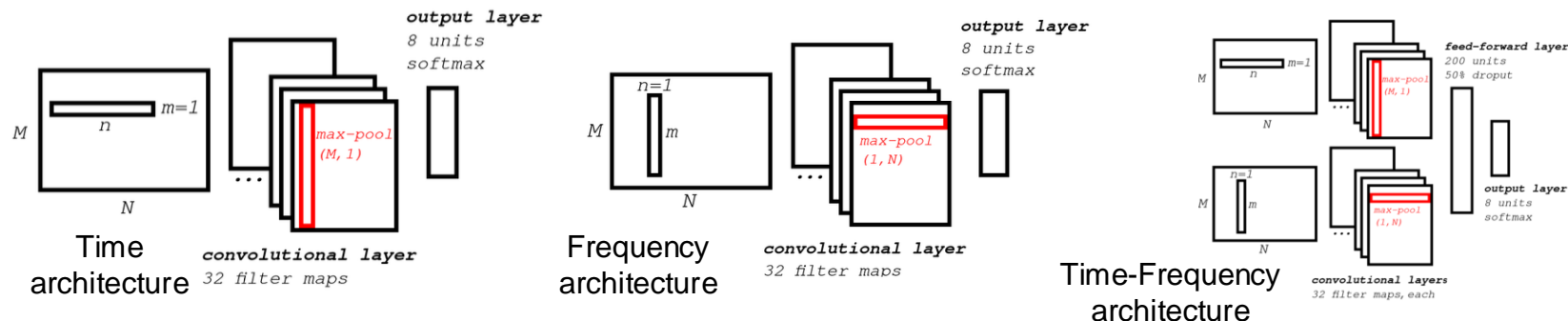
CNN for Music Classification: 2D CNN

- Fully Convolutional Network (FCN)
 - 96-bin mel-spectrogram with 30 second
 - 3x3 filters and **large max-pooling sizes** in time to reduce the temporal dimensionality instead of using temporal summary in the back-end
- Short-Chunk CNN
 - 128-bin mel-spectrogram with 3.69 second (short audio chunk)
 - 3x3 filters and 2x2 max-pooling
 - Summarize the features using global max pooling

FCN-4
Mel-spectrogram (<i>input: 96→1366→1</i>)
Conv 3→3→128
MP (2, 4) (<i>output: 48→341→128</i>)
Conv 3→3→384
MP (4, 5) (<i>output: 24→85→384</i>)
Conv 3→3→768
MP (3, 8) (<i>output: 12→21→768</i>)
Conv 3→3→2048
MP (4, 8) (<i>output: 1→1→2048</i>)
Output 50→1 (sigmoid)

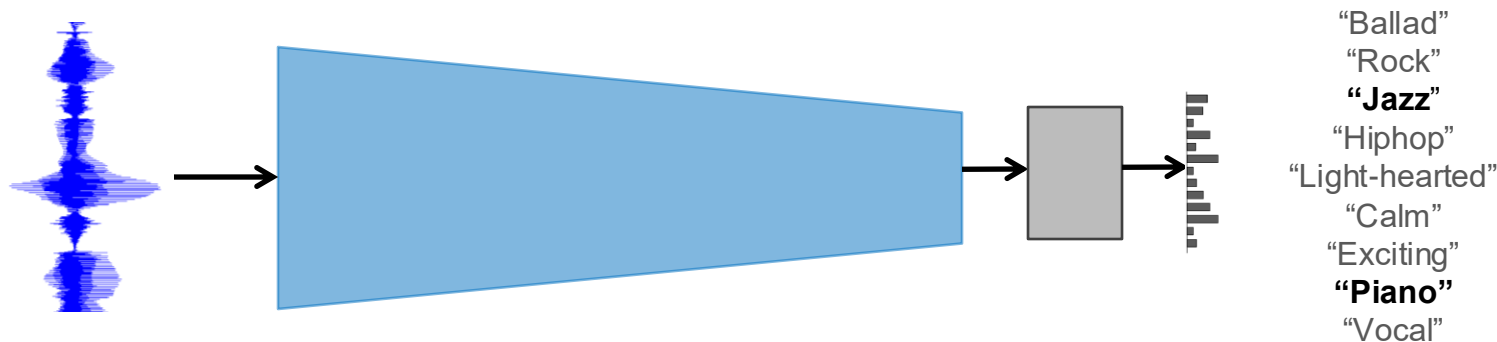
CNN for Music Classification: 2D CNN

- Explore musically-motivated architectures using different “time-frequency” shapes of conv filters in the first layer (“**MusiCNN**”)
 - **Time architecture**: learn rhythm and tempo patterns ($1 \times n$)
 - **Frequency architecture**: learn pitch and timbre patterns ($m \times 1$)
 - **Time-Frequency architecture**: combine the two
 - Performance: TF > T > F on the ballroom music dataset
 - Can be **extended to multiple parallel branches** (e.g. 1×9 , 3×3 , 9×1)



CNN-based Model for Music Classification

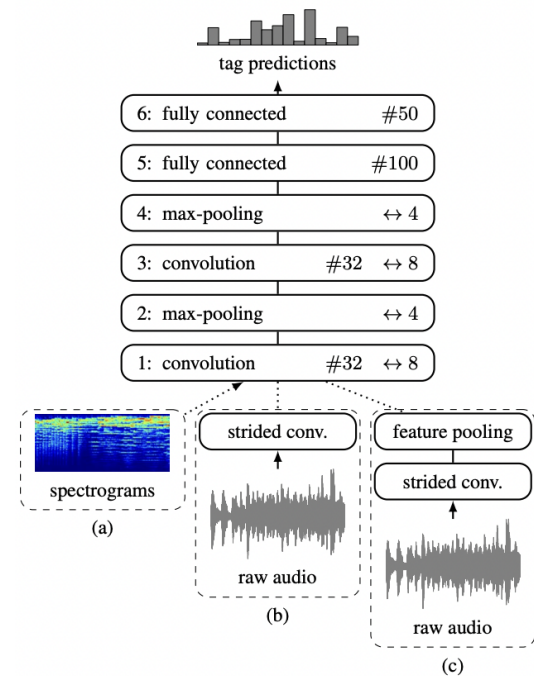
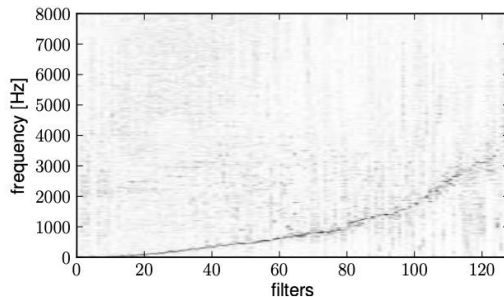
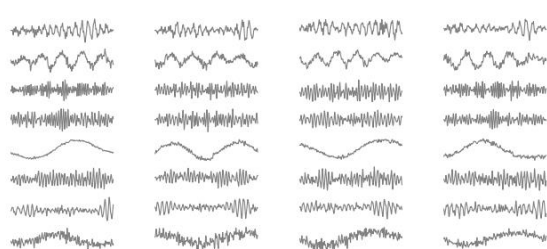
- How about learning the time-frequency representation (or filter banks) directly from raw waveforms?



- No need to tuning STFT and mel parameters
- No need to store the preprocessed mel-spectrogram on disk space

Learning Filter banks in the Frame Level

- 1D CNN takes raw waveforms directly as input
 - The filter size in the first conv layer is set to “frame-level” (256, 512, 1024 samples)
- They learn “log-like” filter banks
 - The peak frequency exponentially increases
- However, failed to outperform mel-specrogram

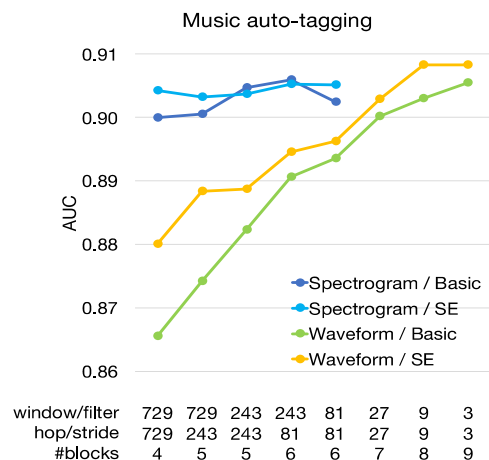
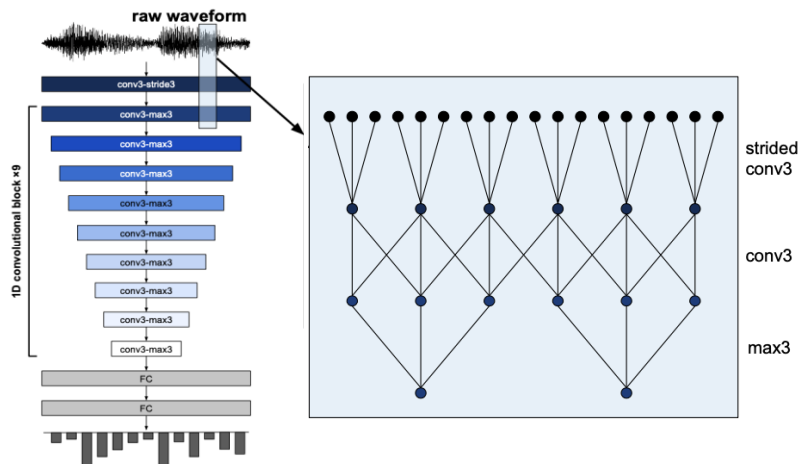


length	stride	AUC (spectrograms)	AUC (raw audio)
1024	1024	0.8690	0.8366
1024	512	0.8726	0.8365
512	512	0.8793	0.8386
512	256	0.8793	0.8408
256	256	0.8815	0.8487



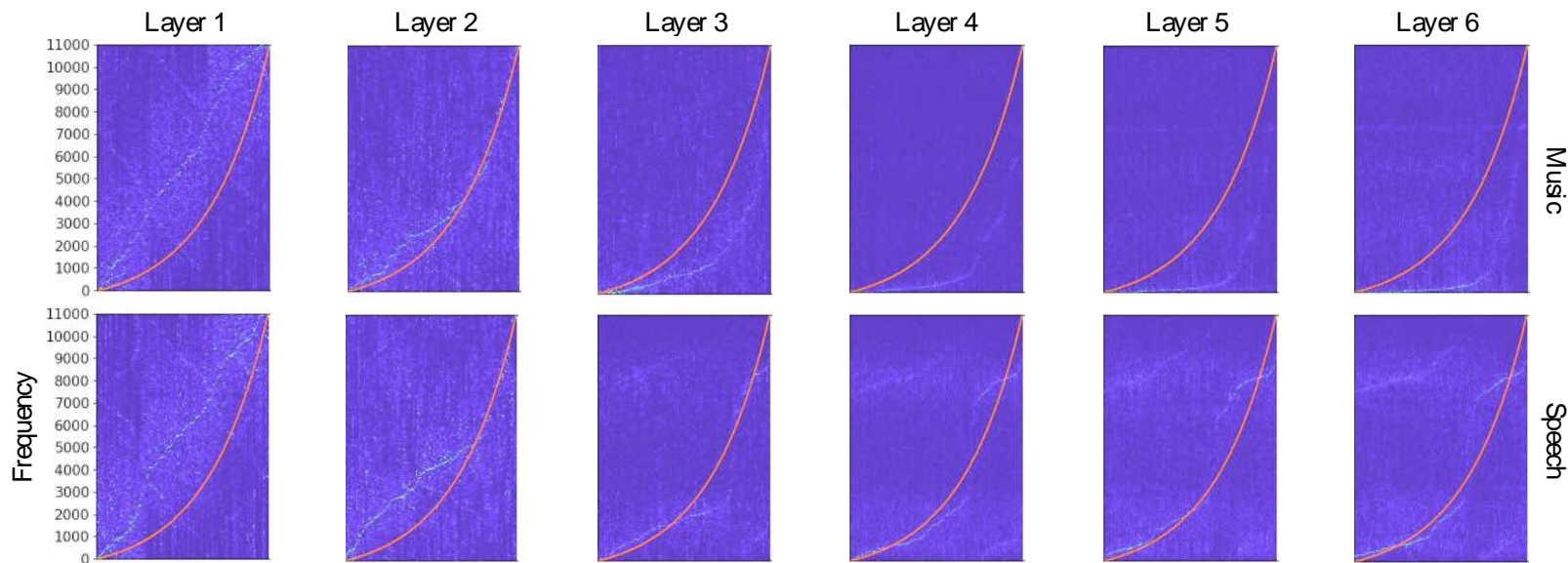
Learning Filter banks in the Sample Level (SampleCNN)

- Use 1D convolution blocks with very small size of filters
 - The filter size is 3 x 1 throughout the entire layers: “1D VGGNet”
 - The first conv layer has a stride of 3 and the rest have max pooling of 3
 - As the filter size becomes smaller, the model gets deeper, achieving better performance increases



Learning Filter banks: SampleCNN

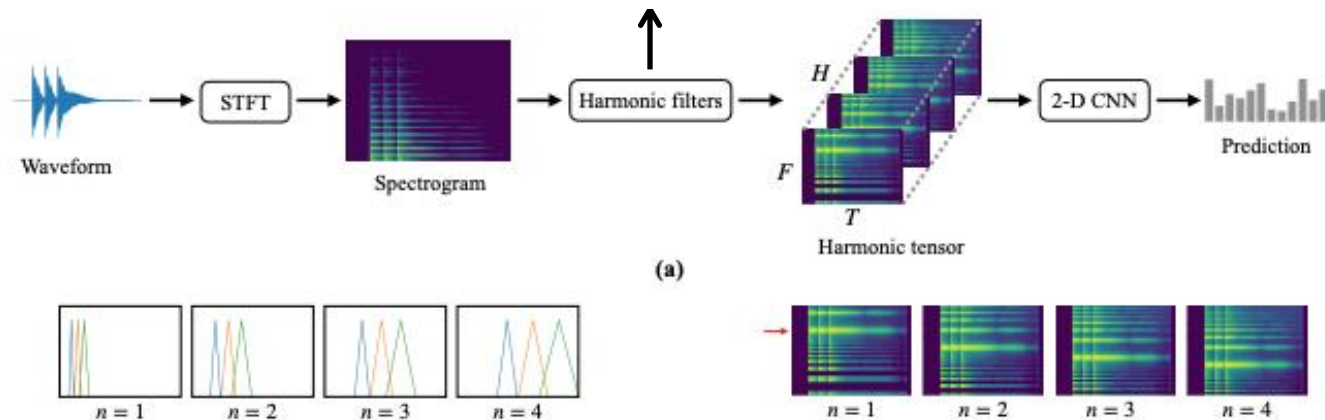
- The learned filters have a trend of log scale
 - When the audio input is speech, the trend is more similar to the mel scale



Learning Filter banks: HarmonicCNN

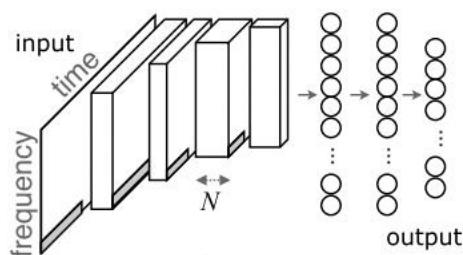
- Partially learnable filter bank
 - Use spectrogram and learn the shape of triangular in mel filter banks
 - Learn a set of harmonically scaled filter banks
 - Achieved better performance than SampleCNN

$$\Lambda_n(f; f_c, \alpha, \beta, Q) = \left[1 - \frac{2|f - n \cdot f_c|}{(n \cdot \alpha f_c + \beta)/Q} \right]_+ \quad a \text{ and } \beta \text{ determine the width of the triangle}$$

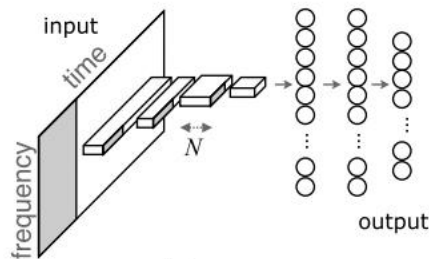


CNN for Music Classification: 2D CNN

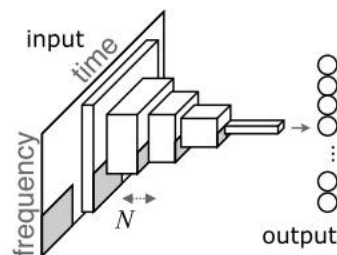
- Use RNN for temporal summary in the back-end
 - Compared to various CNN models (1D CNN, 2D CNN)
 - The CRNN model slightly outperforms the CNN models but it is slower



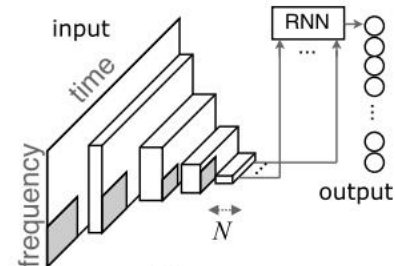
(a) k1c2



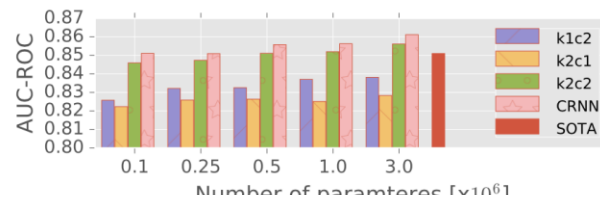
(b) k2c1



(c) k2c2

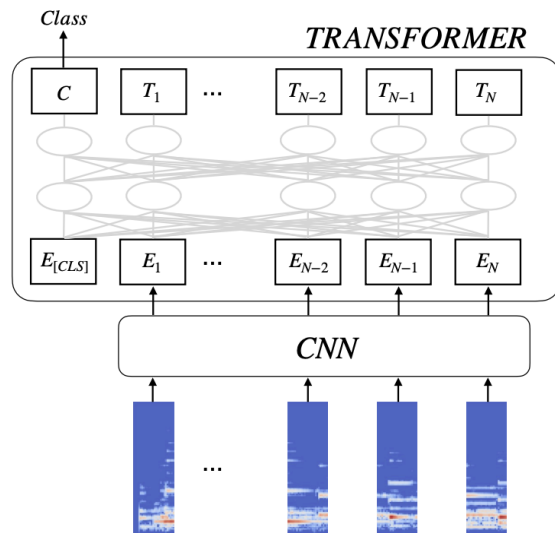


(d) CRNN



Summary by Transformer

- The back-end part is summarized with a transformer module
 - Learning the long-term dependency with the self attentions layers
 - The special token embedding (CLS token) is used for classification



Dataset: Genre Classification

- GTZAN: <http://marsyas.info/downloads/datasets.html>

US Pop Genre Classification

Blues
Jazz
Country/Western
Baroque
Classical
Romantic
Electronica
Hip-Hop
Rock
HardRock/Metal

Latin Genre Classification

Axe
Bachata
Bolero
Forro
Gaucha
Merengue
Pagode
Salsa
Sertaneja
Tango

K-pop Classification

Ballad
Dance
Folk
Hip-hop
R&B
Rock
Trot

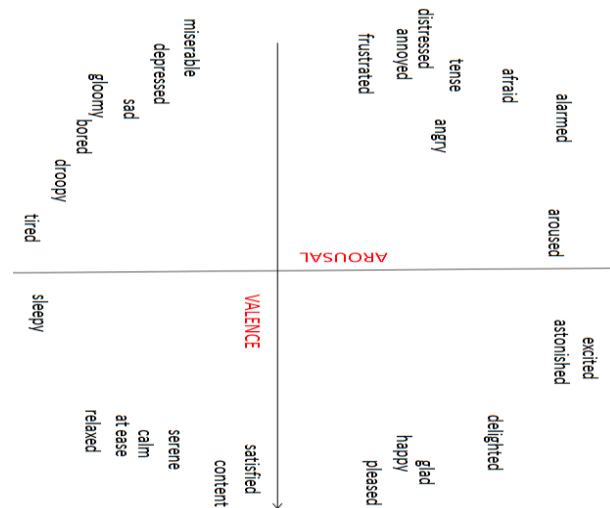
[https://www.music-ir.org/mirex/wiki/2020:Audio Classification \(Train/Test\) Tasks](https://www.music-ir.org/mirex/wiki/2020:Audio_Classification_(Train/Test)_Tasks)

Dataset: Musical Instrument Recognition

- Single monophonic instrument
 - Nsynth: <https://magenta.tensorflow.org/datasets/nsynth>
 - TinySol: <https://zenodo.org/records/3685367>
- Pre-dominant instrument
 - IRMAS: <https://zenodo.org/records/1290750>
- Multiple instruments:
 - Openmic-2018: <https://github.com/cosmir/openmic-2018>

Dataset: Mood Classification/Regression

- Collections: https://github.com/juansgomez87/datasets_emotion
- Classification:
 - categorical words
- Regression: (arousal, valence)
 - continuous values



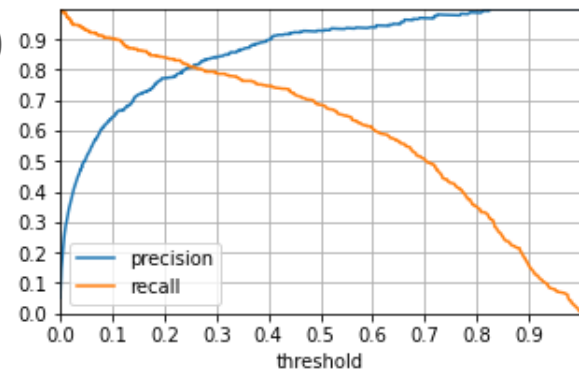
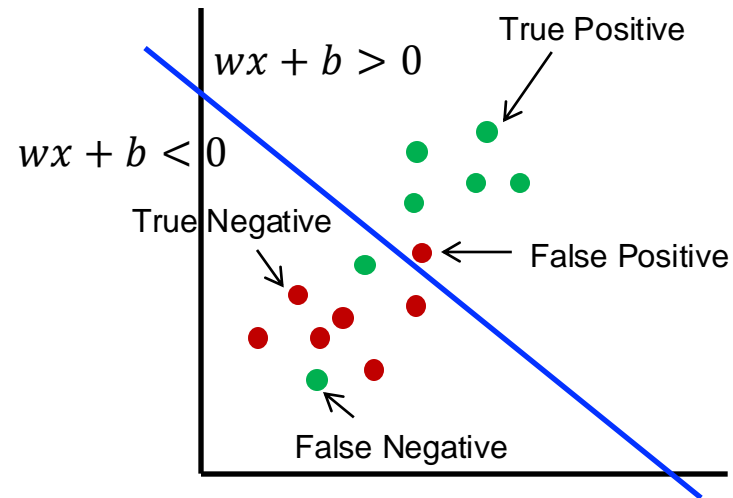
Russel's circumplex model of affect
"Arousal-Valence" 2D space

Dataset: Music Tagging and more

- Music Tagging
 - MagnaTagaTune (MTT): <https://mirg.city.ac.uk/codeapps/the-magnatagatune-dataset>
 - Million Song Dataset (MSD): <http://millionsongdataset.com/>
 - Free Music Archive (FMA): <https://github.com/mdeff/fma>
 - MTG-Jamendo: <https://github.com/MTG/mtg-jamendo-dataset>
- General audio classification
 - AudioSet: <https://research.google.com/audioset/>

Evaluation

- Binary classification
 - True Positive (TP), False Positive (FP)
 - True Negative (TN), False Negative (FN)
- Metrics
 - Accuracy = $(TP+TN)/(TP+TN+FP+FN)$
 - Precision = $TP/(TP+FP)$
 - Recall = $TP/(TP+FN)$
 - F1-score = $2 * \text{Precision} * \text{Recall} / (\text{Precision} + \text{Recall})$
 - Harmonic mean of Precision and Recall



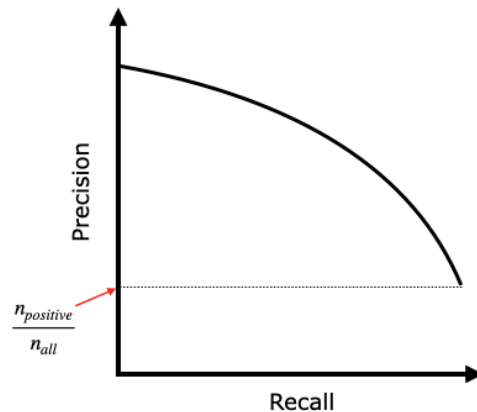
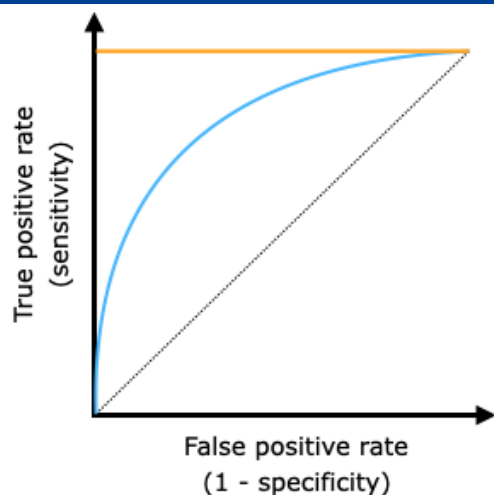
Evaluation

- ROC-AUC

- Area Under Receiver Operating Characteristic Curve
- True positive rate (recall) = $TP/(TP+FN)$
- False positive rate (1-specificity) = $FP/(TN+FP)$
- 0.5 is the lowest value
- Overly optimistic results with imbalanced data

- PR-AUC

- Area under precision-recall curve
- Precision = $TP/(TP+FP)$
- Recall = $TP/(TP+FN)$
- More reliable with imbalanced data



Benchmark of CNN-based Music Classification Models

- Improvement of annotation and retrieval accuracy

Methods	MTAT		MSD		MTG-Jamendo	
	ROC-AUC	PR-AUC	ROC-AUC	PR-AUC	ROC-AUC	PR-AUC
FCN [1]	0.9005	0.4295	0.8744	0.2970	0.8255	0.2801
FCN (with 128 Mel bins)	0.8994	0.4236	0.8742	0.2963	0.8245	0.2792
Musicnn [2]	0.9106	0.4493	0.8803	0.2983	0.8226	0.2713
Musicnn (with 128 Mel bins)	0.9092	0.4546	0.8788	0.3036	0.8275	0.2810
Sample-level [3]	0.9058	0.4422	0.8789	0.2959	0.8208	0.2742
Sample-level + SE [4]	0.9103	0.4520	0.8838	0.3109	0.8233	0.2784
CRNN [6]	0.8722	0.3625	0.8499	0.2469	0.7978	0.2358
CRNN (with 128 Mel bins)	0.8703	0.3601	0.8460	0.2330	0.7984	0.2378
Self-attention [7]	0.9077	0.4445	0.8810	0.3103	0.8261	0.2883
Harmonic CNN [9]	0.9127	0.4611	0.8898	0.3298	0.8322	0.2956
Short-chunk CNN	0.9126	0.4590	0.8883	0.3251	0.8324	0.2976
Short-chunk CNN + Res	0.9129	0.4614	0.8898	0.3280	0.8316	0.2951

Resources

- Pretrained music and audio classification models
 - <https://github.com/minzwon/sota-music-tagging-models>
 - <https://github.com/jordipons/musicnn>
 - <https://github.com/tensorflow/models/tree/master/research/audioset/vggish>
 - <https://github.com/marl/openl3>
- Pretrained model repositories
 - <https://pytorch.org/audio/stable/models.html>
 - <https://huggingface.co/models>