Automatic Music Classification

Presentation for the PAMI Seminar

Andres Fernandez · Goethe Universität Frankfurt · November 2016

1. Outline

2. Models

3. Paper

4. Follow-up and discussion

5. References

Topic and structure

This presentation discusses the problem of automatic music classification by combining basics (based on some tutorial literature) with a current approach (covering a 2016 paper[3]), as follows:

- Present the general problematic and technical background
- Presenting the models used in the paper
- Presenting the paper
- Follow-up, discussion

When needed, the sources are quoted in square brackets (f.e., **[1]**)

Background: automatic classification

The bottom line is to formulate models that, based on some prior knowledge, are able to perform classification on arbitrary samples in a general and effective way. With some caveats:

- If the characteristic features for each class are well known and formulable, they can be directly hard-coded into the model:

```
if (signal.hasInstrument("Bassoon")){ // here it is assumed that
    return "Classes.BORING";          // music with bassoons is
} else { …                            // invariantly boring
```

- For classifications involving also non-trivial features, these can be **inferred** from a given dataset. This is the typical setup for all **supervised learning** problems, and comprises two main strategies[1]:
  - Map the input to a very generic, high-dimensional space and learn the boundary there (f.e. SVMs).
  - Learn the different layers of the mapping itself (multi-layer NNs), which gives up convexity but allows flexibility.

Background: supervised learning

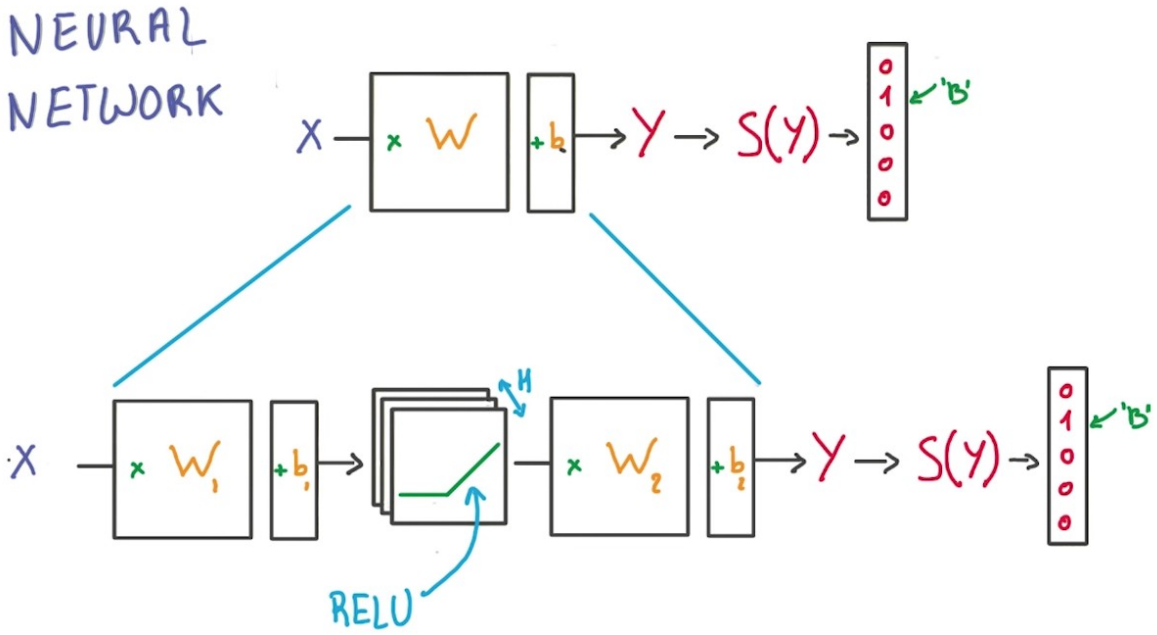
From **[2]**:

> The task of supervised learning is: given a **training set** (X, y) of m input-output pairs $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \cdots (x^{(m)}, y^{(m)})$, where each $y^{(j)}$ was generated by an unknown function $y = f(x)$, discover a function h that approximates the *true function* f.

- Hypothesis function: if classifying, the output of h is discrete.
- Parametric: h combines the input x with a fixed set of to-be-learned parameters.
- Evaluating: The quality of h is measured by a cost function J that compares each hypothesis with the corresponding $y^{(i)}$ label, and punishes big differences.
- Learning: The learning process consists in minimizing J for the entire dataset by changing the parameters. If J is convex, this can be analytically done. If not, the gradient descent family of algorithms can be used.

Background: neural networks

Some supervised-learning models are all-purpose: for example, 3-layer MLP with RELU+Softmax activators, trained with backpropagation and gradient descent on the cross-entropy cost function (see **[4]** and **[5]**):



$$h_c(x, \Theta^{(1)}, \Theta^{(2)}, b^{(1)}, b^{(2)}) = max(softmax(c, \Theta^{(2)}\ (relu(\Theta^{(1)}x + b^{(1)})) + b^{(2)}))$$

$$cost(\Theta, X, y) = \frac{1}{N}\sum_{i=i}^{N}\{-log(S(y_i, h_{y_i}(X_{(i,.)})))\} \qquad softmax(c, x_1, ..., x_N) = \frac{e^{x_c}}{\sum_{i=i}^{N} e^{x_i}}$$

$$\frac{\partial \mathcal{L}}{\partial z_j^{(i)}} = \sum_{c=1}^{C} \frac{\partial \mathcal{L}}{\partial S(z^{(i)})_c} \frac{\partial S(z^{(i)})_c}{\partial z_j^{(i)}} = S(z^{(i)})_j - \vec{y}_j^{(i)} \quad 1 \leq j \leq C, 1 \leq i \leq m, \quad z^{(i)} := \Theta^T x^{(i)}$$

Background: optimization objective

Multi-layer NNs like the shown before are good at classifying data. They are regarded as **universal approximators**, because, if given enough size and time, they are able to capture any non-linear relations, and to approximate any kind of continuous function, as stated in the universal approximation theorem[8].

The caveat lies in the *given enough size and time* part: it is rare that a setup can count on enough of both. In this context, the so-called **no free lunch theorem**[9] states that, in order to optimize a learning process, some assumptions about the dataset have to be (effectively) done.

Being able to successfully formulate and implement assumptions in a model can be interesting in order to learn from the data, and may also be very valuable if optimization is needed. Due to the flexibility of their approach, artificial neural networks comprise many different architectures, that enable different forms of optimization.

Topic: update

Now it is possible to update the goal of this presentation, namely to discuss

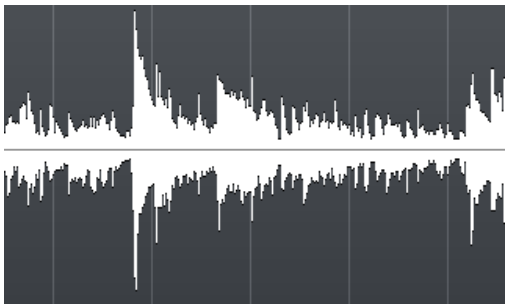
Supervised learning models that are able to **optimally** classify **non-trivial** music samples based on some **labeled dataset**

Which arises the following questions:

- How to conveniently represent the input data?
- Which assumptions can be made on the dataset?
- Which models profit at best from those assumptions and how?

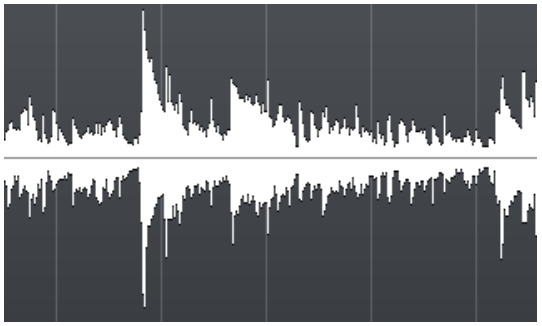
Representation of the data

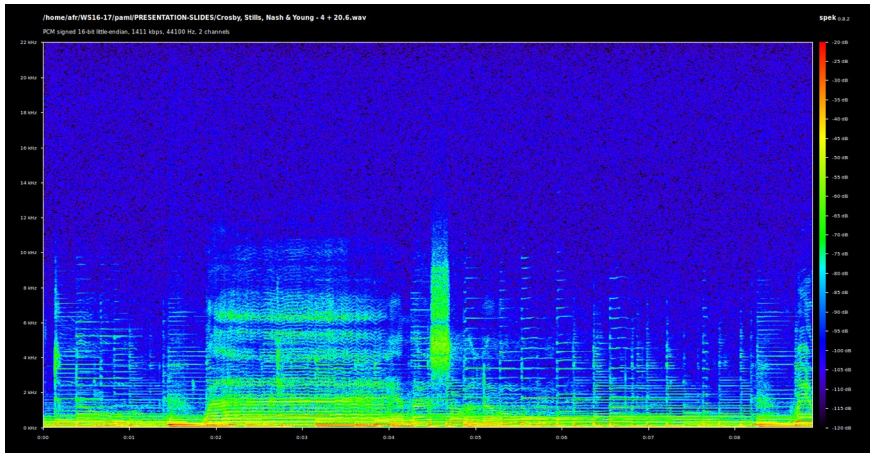
Music can be modeled as a sequence of ($\boldsymbol{x}^{(1)}$, $\boldsymbol{x}^{(2)}$, ... , $\boldsymbol{x}^{(\tau)}$) D-dimensional vectors:



$$\left(\begin{bmatrix} x_1^{(1)} \\ \dots \\ x_D^{(1)} \end{bmatrix}, \begin{bmatrix} x_1^{(1)} \\ \dots \\ x_D^{(1)} \end{bmatrix}, \dots, \begin{bmatrix} x_1^{(\tau)} \\ \dots \\ x_D^{(\tau)} \end{bmatrix} \right)$$

Whereas the superscript reflects the quantization of the time domain (a sample). This is is pretty much the way uncompressed recordings are stored in digital devices (for example the *.wav* format). Music can also be represented on the frequency domain with help of the Fourier-Transformation, without any loss of information.





Assumptions on the data

Firstly, it is assumed that the dataset is meaningfully and correctly labeled. This means that each different class present some characteristic, general, **invariant features** thorough its data.

Depending on how does the dataset look like, the specific features may vary strongly, as well as the extent in which they can be assumed. Some examples can be:

- Non-fixed length: not all music samples must have equal size
- Phase shifting: the whole signal can be delayed
- Frequency shifting: the whole signal can be transposed
- Redundancy: similar fragments can appear in different places
- Motivic base: different sources can play the same music

An empirical approach to the assumptions

Other assumptions may be done, like the existence of an underlying rhythmic/melodic/harmonic structure, but may also be more difficult to generalize. The usual “no free lunch” strategy, as with regularization, is to choose a model with “loose” assumptions that help it to converge faster to a good solution, but letting the learning do the close-up job, and then evaluate them empirically with some sort of cross-validation, as is the case of the paper analyzed in the third part of this presentation, involving CNNs and RNNs:

> “We compare CRNN with three CNN structures that have been used for music tagging while controlling the number of parameters with respect to their performance and training time per sample. Overall, we found that CRNNs show a strong performance with respect to the number of parameter and training time, indicating the effectiveness of its hybrid structure In music feature extraction and feature summarisation.” [3]

Before getting into the paper details, let's briefly cover CNNs and RNNs to see why this could be possible.

1. Outline

2. Models

3. Paper

4. Follow-up and discussion

5. References

Parameter sharing

Both convolutional and recurrent neural networks have something in common, that differentiates them from the vanilla, fully-connected ones: their use of **parameter sharing**. In a traditional parametrical fully-connected NN model,

- All tensors thorough each layer have the same size
- Each parameter represents exactly one connection

On the other side, sharing the parameters “refers to using the same parameter for more than one function in a model”. This means that:

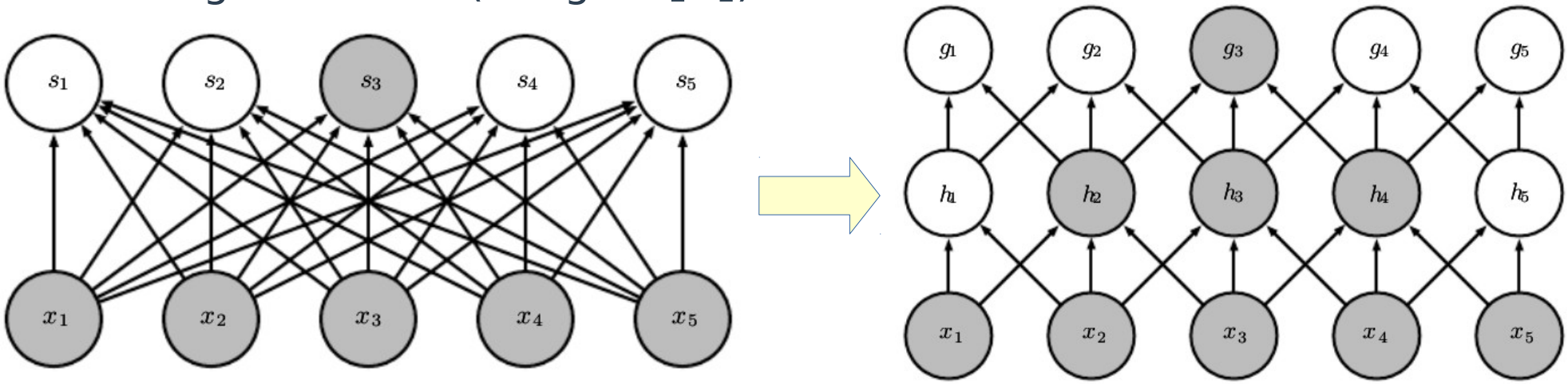
- Modularization: The model will have components (called *kernels* or *patches*) that are applied several times on the same input. The parameters of each patch (also called *tied weights*) are shared across the regions of the input where they are applied.
- Further customization: The model will assume similarity in the domain in which the parameters are shared. This domain can be highly customized. Furthermore, different patches can be applied on the same domain.

Parameter sharing

In conclusion,

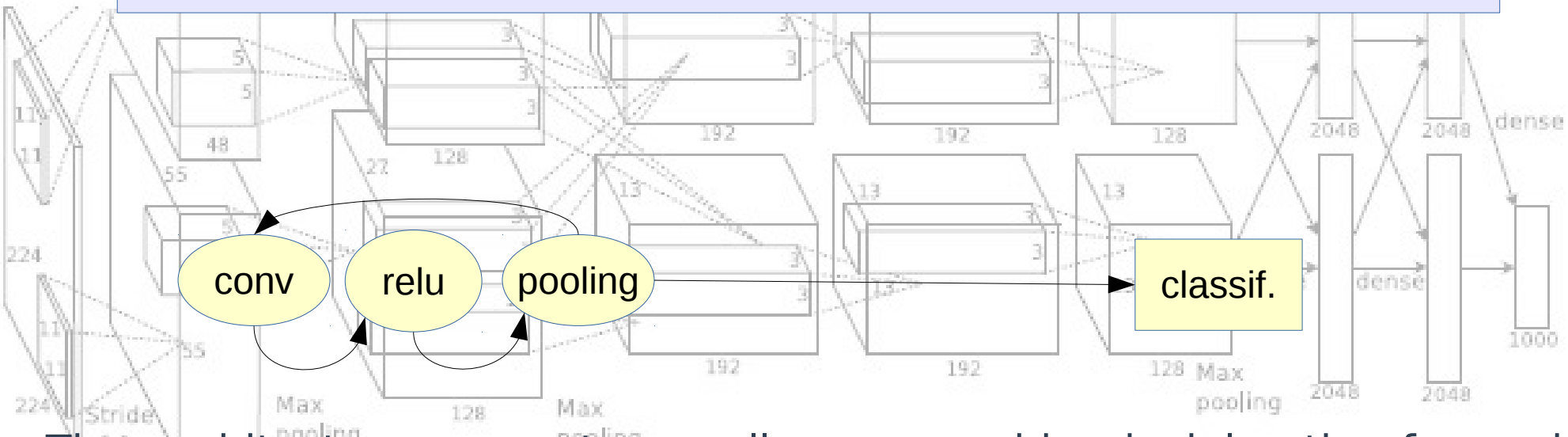
Instead of “asking for every detail”, designing and applying parameter sharing forces the model to “design specific questions”, and “which domain should they be asked in”.

This implies that the relations between the input features are represented in a much more “sparse” way, which allows forward- and backpropagations to be less CPU- and memory- expensive, but on the other hand it has to be done “right” in order for it to provide meaningful results (Images: **[1]**).



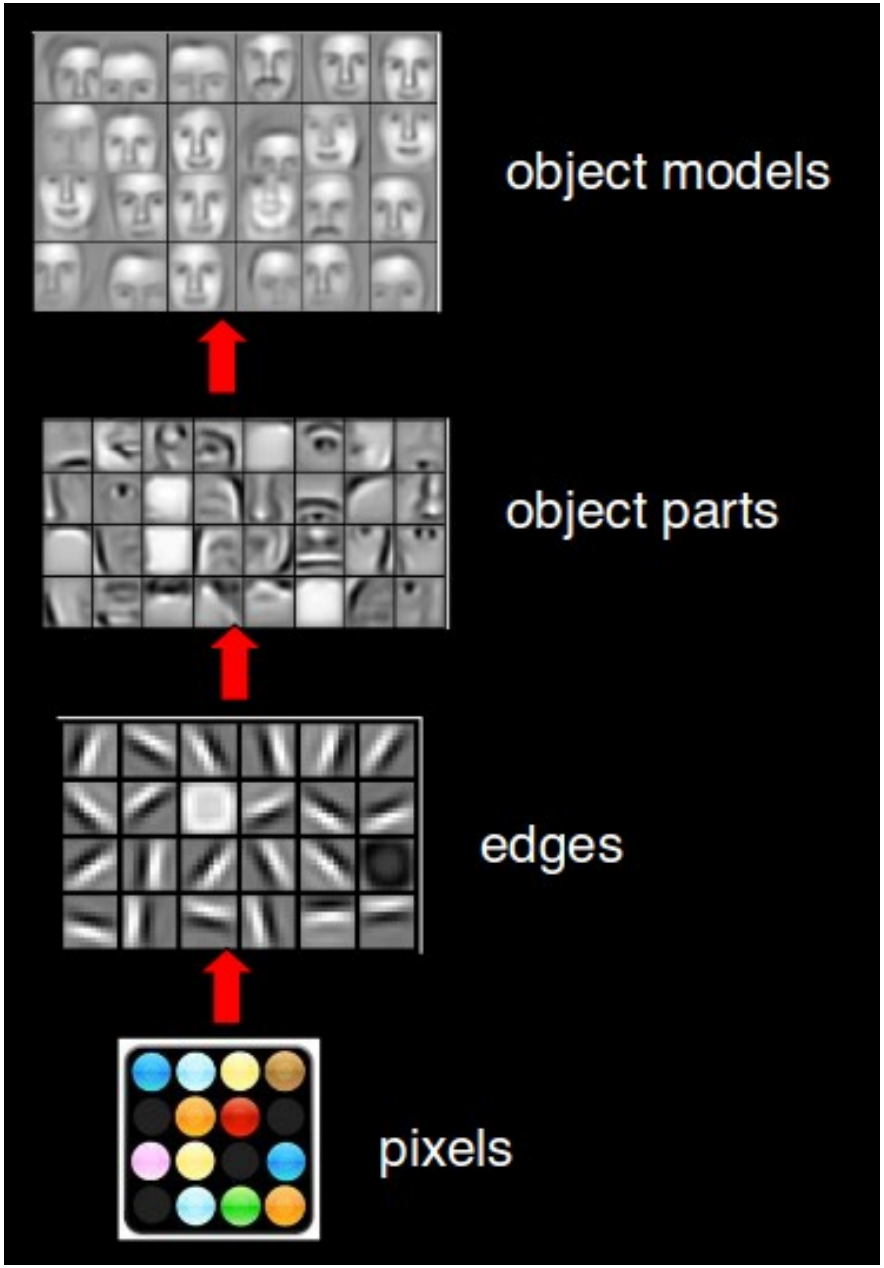
CNNs: architecture

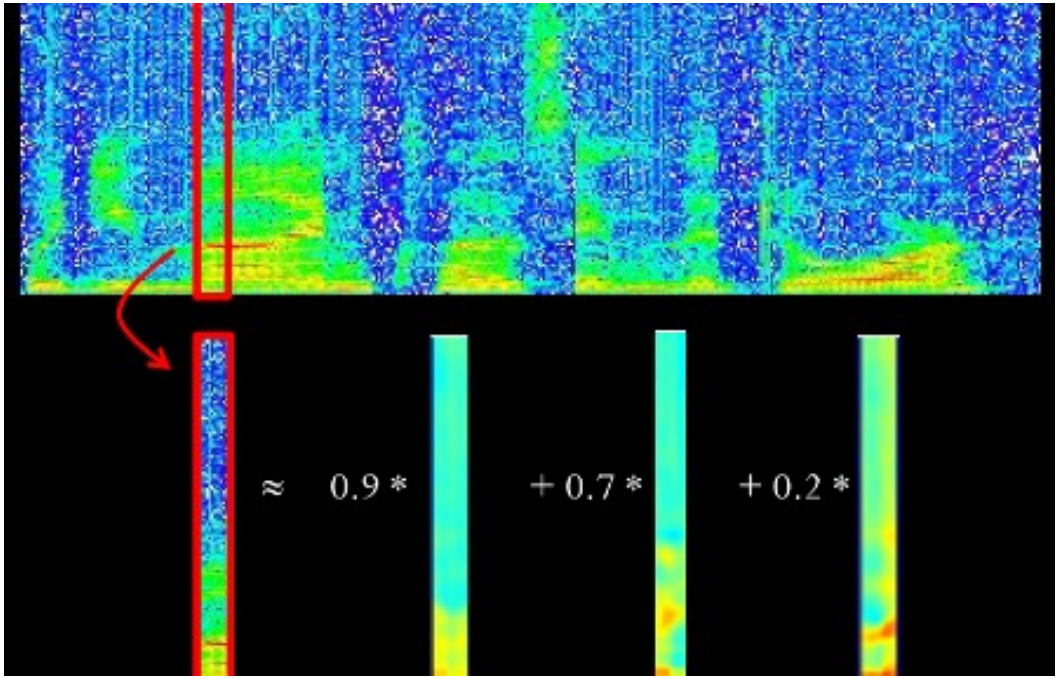
"Convolutional networks are simply neural networks that use Convolution in place of general matrix multiplication in at least One of their layers"[1]



The architecture presents usually a pyramid principle: the forward propagation transitions from broad, "flat" representations to narrow, "deeper" ones, or equivalently: from few simple, extensive patches to many different, complexer, broader ones. This "deep", learned representation is fed to a classifier on the top (usually a fully-connected NN). Background image: *AlexNet*

CNNs: example images from [10]



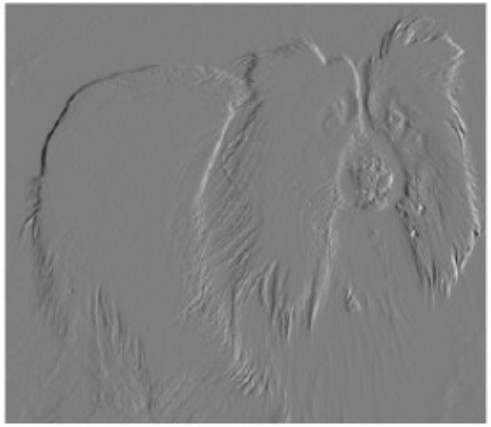


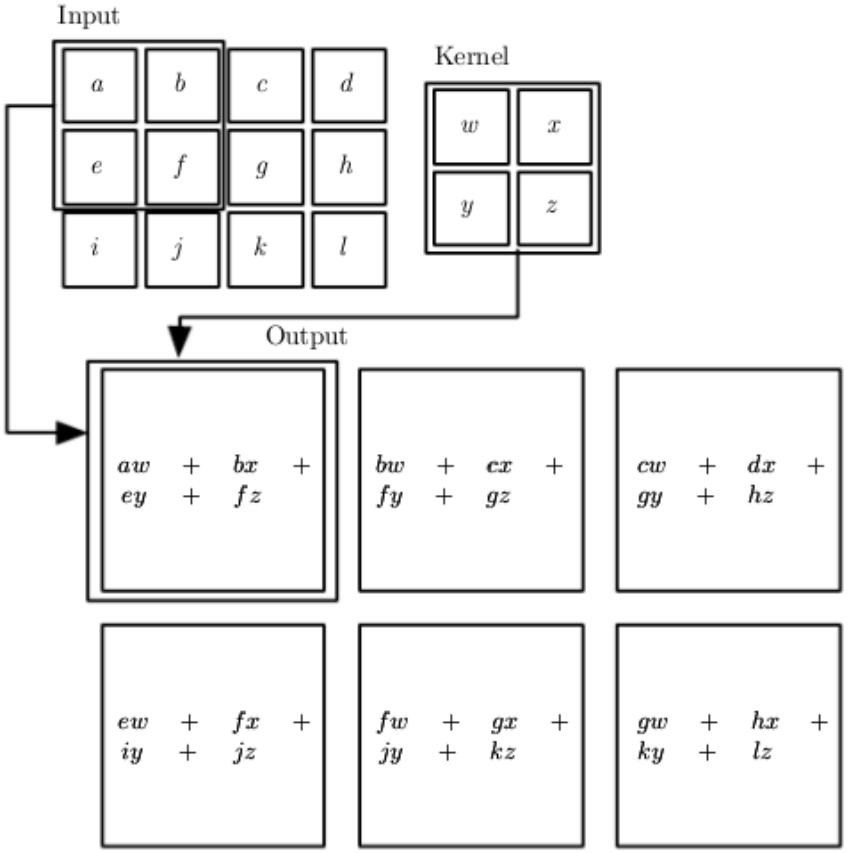
CNNs: Convolution [1]

The convolution is a mathematical function that operates on two signals. The 1-dimensional, discrete convolution is defined as follows:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$$

As the dot product, it can be seen as an "affinity filter" between the two signals *x* and *w*, but with an extra parameter *t*, which can be given many different meanings: if the axis *a* is the time, *t* represents a time delay. If *a* is space, *t* represents distance. For instance, the convolution can be extended to more than one dimension:

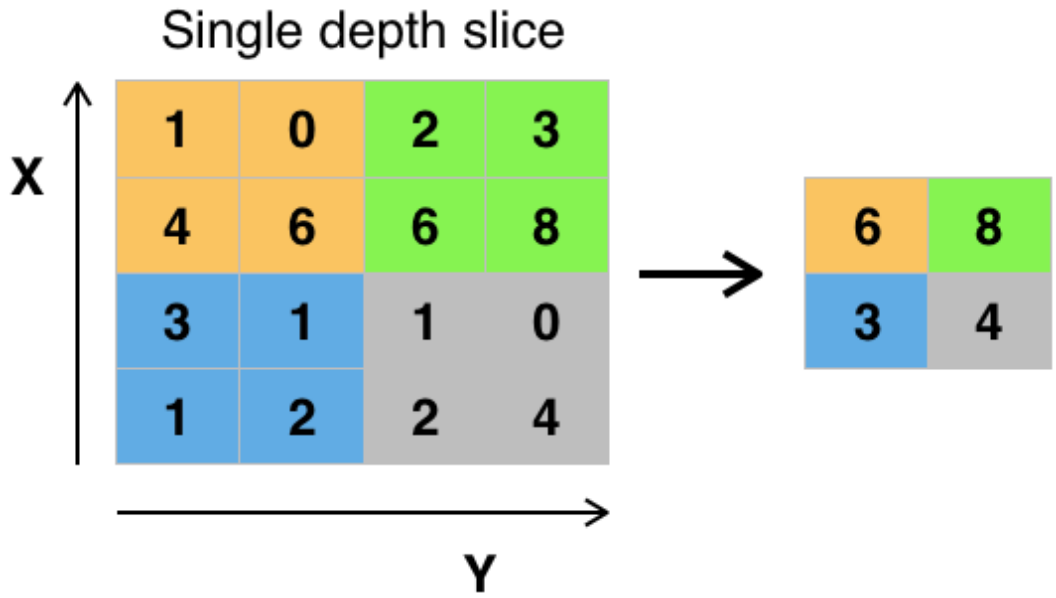




CNNs: Pooling [1]

In comparison to convolution, pooling is a rather intuitive operation: it replaces a region of “nearby” values with a single value that is representative of them. For example:

- Max pooling: the biggest of them (see figure for a 2-D example)
- Average pooling
- Weighted pooling
- ...



This operation enables the model to learn some invariances from the data, like translation, rotation... (see **[11]**)

CNNs: Learning and conclusions

The cost function and optimization objective of CNNs depend on the exact architecture used, and is not in the scope of this presentation (see **[1,p. 358]** for more details on that). Here, it will suffice to say that CNNs are trained with the gradient descent + backpropagation combo, in a similar fashion as the vanilla NNs.

Regarding the assumptions that could be done on the music data, some features of this model could prove useful:

- Ability to process inputs of different sizes
- Ability to learn translation invariance (time→delay, spectrum→ transp.)
- Efficient use of parameters for detecting redundancies

RNNs: architecture

As CNNs, RNNs also incorporate parameter sharing:

> “Parameter sharing makes it possible to extend and apply the model to examples of different forms (different lengths, here) and generalize across them. If we had separate parameters for each value of the time index, we could not generalize to sequence lengths not seen during training, nor share statistical strength across different sequence lengths and across different positions in time. Such sharing is particularly Important when a specific piece of information can occur at multiple positions within the sequence. For example, consider the two sentences: »I went to Nepal in 2009« and »In 2009, I went to Nepal.«” [1]

A 1-D convolution would have also this properties, but the parameters would be shared in a hierarchically similar way thorough the sequence. **RNNs involve a recursion that enables parameter sharing across a deep structure**[1]. In pseudocode:

```
def cnn1d(dataset, weights):
   return [f(x, weights)
           for x in dataset]
```

```
def rnn(t, x=x0, w):
    if (t is 0): return f(x,w)
    else: return rnn(t-1,f(x,w),w)
```

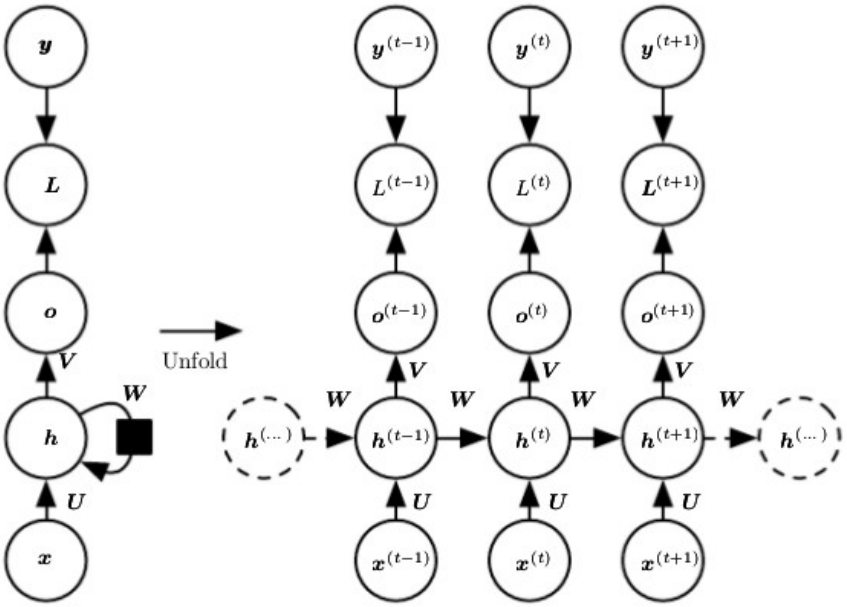
RNNs: three flavors

All NNs involving such a recursion are considered RNNs. Three different flavors[1]:

- Recurrent networks that produce an output at each time step and have recurrent connections between hidden units.
- Recurrent networks that produce an output at each time step and have recurrent connections only from the output at one time step to the hidden units at the next time step.
- Recurrent networks with recurrent connections between hidden units, that read an entire sequence and then produce a single output.

From the three, the first one is the most complex, as it can perform every computation a universal Turing Machine does. The other two are more limited, optimized versions: lacking the hidden-to-hidden recurrence, but having therefore a parallelizable learning algorithm.

RNNs: hidden-to-hidden hypothesis and cost[1]



$$U \hat{=} \text{ input-to-hidden}$$
$$V \hat{=} \text{ hidden-to-output}$$
$$W \hat{=} \text{ hidden-to-hidden}$$

$$\begin{aligned} \boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}) \\ \boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \\ \hat{\boldsymbol{y}}^{(t)} &= \text{softmax}(\boldsymbol{o}^{(t)}) \end{aligned}$$

- Forward-propagation: given an initial state $h^{(0)}$, and the above mentioned parameters, update until reaching the desired *t:*

$$\boldsymbol{a}^{(t)} = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}$$

- The total loss would be the accumulated loss from 0 to *t*. For the cross-entropy:

$$L\left(\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(\tau)}\}, \{\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(\tau)}\}\right)$$
$$= \sum_t L^{(t)} = -\sum_t \log p_{\text{model}}\left(y^{(t)} \mid \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t)}\}\right)$$

RNNs learning: the BPTT[1]

BPTT stands for Back-Propagation Through Time. If regarding the graph in its unfolded variant, it becomes clear its analogy to a feedforward NN, so no specialized algorithms are needed: just gradient descent with backpropagation. In fact, for the cross-entropy cost function, the output equation remains the same, simply substract the predicted value from its label:

$$(\nabla_{\boldsymbol{o}^{(t)}} L)_i = \frac{\partial L}{\partial o_i^{(t)}} = \frac{\partial L}{\partial L^{(t)}} \frac{\partial L^{(t)}}{\partial o_i^{(t)}} = \hat{y}_i^{(t)} - \mathbf{1}_{i,y^{(t)}}$$

After that, it comes the *h* layer:

$$\nabla_{\boldsymbol{h}^{(\tau)}} L = (\nabla_{\boldsymbol{o}^{(\tau)}} L) \frac{\partial \boldsymbol{o}^{(\tau)}}{\partial \boldsymbol{h}^{(\tau)}} = (\nabla_{\boldsymbol{o}^{(\tau)}} L) \boldsymbol{V}$$

And so on for every layer, in every cycle, until reaching the origin, or some other imposed limit. See **[1, p. 386]** for more details on the BPTT.

RNNs: conclusions

In comparison with CNNs, RNNs can also process inputs of different sizes, and their very deep parameter sharing structure allows them to capture features that express themselves in distant samples, and therefore efficiently learn distant relations with very few parameters.

On the other hand, the high amount of correlated parameters makes the training not only computational expensive, but also numerically unstable. For this different solutions, like LSTMs or gradient normalizing, can be used.

1. Outline

2. Models

3. Paper

4. Follow-up and discussion

5. References

Overview

CONVOLUTIONAL RECURRENT NEURAL NETWORKS FOR MUSIC CLASSIFICATION (Choi, Fazekas, Sandler, Cho : 2016)

- This paper tackles the problem of finding an optimal architecture for "music tagging" in an empirical way: namely, to predict genres, moods, instruments, and eras on the *Million Song* dataset.
- Here it is important not only the performance, but the proportion between performance and number of parameters, implying that the chosen parameters represent better the invariant features if this ratio rises.
- For that sake, the paper describes different existing models (*k1c2*, *k2c1* and *k2c2*), and the proposed one (*CRNN*). They also describe the conditions which the models were trained in, and reports their performance, "measured on the test set and by AUC-ROC (Area Under Receiver Operating Characteristic Curve)".
- The conclusions simply state that the proposed model performs better on the chosen experimental setup.

Model conception

The reflexions on the model parts and how could they contribute to the general task can be found in the introduction, and match the general expectations described in this presentation:

> "In CRNNs, CNNs and RNNs play the roles of feature extractor and temporal summarizer, respectively. Adopting an **RNN** enables the networks to **take the global structure into account** while local features are extracted by the remaining convolutional layers. RNNs are more flexible in selecting how to summarize the local features than CNNs which are rather static. This flexibility can be helpful because some of the tags (e.g., mood tags) may be affected by the global structure while other tags such as instruments can be affected by local and short-segment information" [3]

Also, all the architectures "are equipped with **identical optimization** techniques and activation functions, for a **correct comparison**". But "exceptionally", the CRNN has a **weaker dropout** (0.1) "between conv. layers **to prevent overfitting** of the RNN layers".

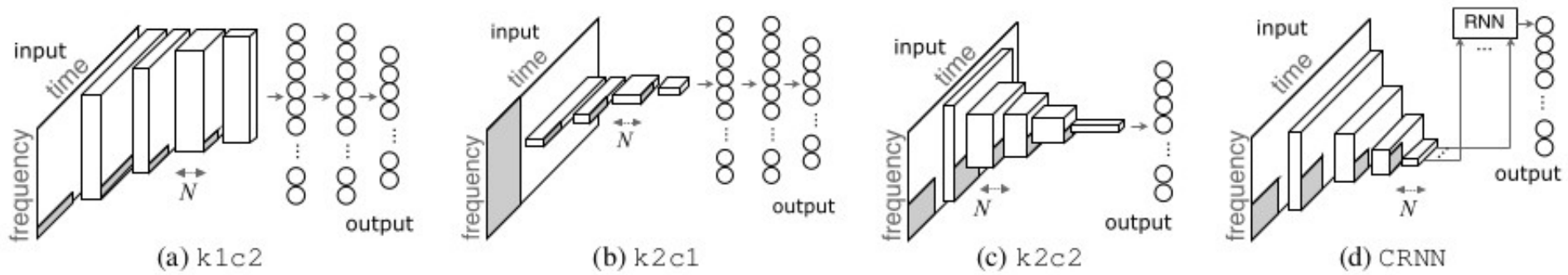
- → This affirmations are made on an empirical basis.

Models

- **k1c2**: 4 convolutional layers (1D-kern,2D-conv, max pool.), followed by 2 fully-connected layers.
- **k2c1**: 5 convolutional layers (2D-kern, 1D-conv, max pool.), followed by 2 fully-connected layers.
- **k2c2**: 5 convolutional layers (2D-kern, 2D-conv, max pool.). Fully convolutional.
- **CRRN**: 2 recursive layers (using gated recurrent units) on the top of 4 convolutional layers (2D-kern, 2D-conv, max pool.).

Regarding proportions in the CRRN:

> "the widths are determined based on preliminary experiments which showed the relative importance of the numbers of the feature maps of convolutional layers over the number of hidden units in RNNs" [3]



Experiments

- **Dataset**: The *Million Song* dataset (201k training, 12k validation), with *last.fm* tags (genres, moods, instruments, eras). Class balance: from about 53k (rock) to 1.2k (happy).
- **Preprocessing**: The songs are trimmed, downsampled, and fourier-transformed (represented with log-amplitude).
- **Cost function and training**: Trained with batch gradient descent with ADAM and backpropagation on the $\log_2$ cross-entropy. Results measured as AUC-ROC (a sort of multi-class F1-Score) on the test set, with early stopping when improvement on the validation set stops.
- **Results**: CRRN>k2c2>k1c2>k2c1 consistently, for all different model sizes (number of trained parameters, see figure). Whereas the CRRN is slower to train, therefore a **memory-speed tradeoff** is observed.

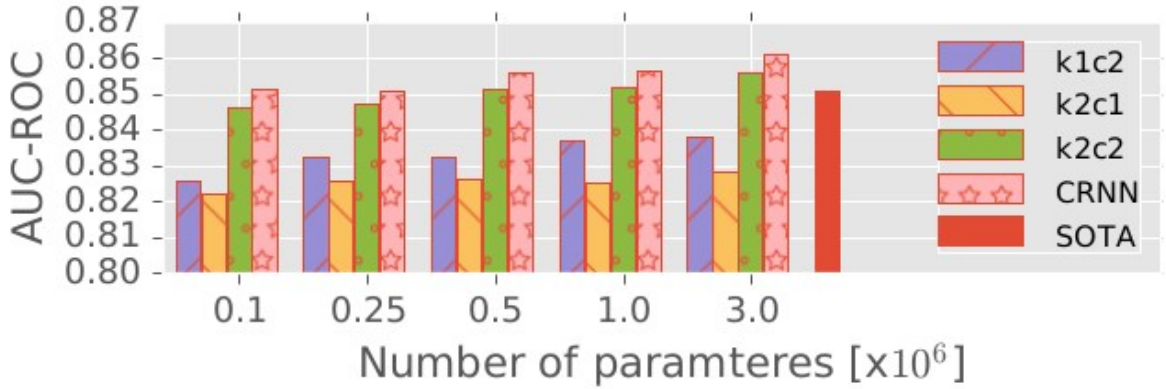


Fig. 2: AUCs for the three structures with $\{0.1, 0.25, 0.5, 1.0, 3.0\} \times 10^6$ parameters. The AUC of SOTA is .851 [2].

1. Outline

2. Models

3. Paper

4. Follow-up and discussion

5. References

Uncovered topics

In a further discussion, following topics may be covered:

- Vanishing/Exploding RNN gradients → LSTM design (video)
- Structured explanation on how the models fullfill assumptions
- Importance of an “educated guess” regarding assumptions
- Further questions?

1. Outline

2. Models

3. Paper

4. Follow-up and discussion

5. References

Uncovered topics

1. Goodfellow, Bengio and Courville (2016): *The Deep Learning Book*
2. Russel and Norvig (2009): *Artificial Intelligence: A Modern Approach*
3. Choi, Fazekas, Sandler and Cho (2016): *CRRNs For Music Classification*
4. Chakraboty, Vanhoucke (2016, video): *Linear Models are Limited*
5. Chakraboty, Vanhoucke (2016, video): *Network of ReLUs*
6. Chakraboty, Vanhoucke (2016, video): *Backprop through time*
7. Chakraboty, Vanhoucke (2016, video): *Vanishing/Exploding gradients*
8. Cybenko (1989): *Approximations by Superimposition of Sigmoidals*
9. Wolpert, Macready (1995): *No Free Lunch Theorems for Search*
10. Andrew Ng (2015, slides): *Unsupervised Feature and Deep Learning*
11. Yann LeCun(1998, webpage): http://yann.lecun.com/exdb/lenet/

Thank you!