Deep learning Audio Classification project

Ferdia Fagan (163 72803)

Abstract

This paper is aimed at being a helpful introduction learning paper for music classification. I will design and explain several different architectures, make some comparisons, as well as what I think the reasoning behind some of these choices in architectures architecture are. [1].

introduction

Firstly, lets look at the data. Audio data is an event (compared to visual data, which could be described as an object). It exists at one moment in time, and then it passes by (i.e. it is highly serial) [2]. Sampling is a way of converting real world Audio from a continuous analog waveform into a digital analog waveform [3]. To choose a sample rate, we must keep in mind Nyquist theorem: which gives us the rule that whatever the maximum frequency is of our bandwidth of interest, we must choose a sample rate greater than twice this frequency. Music is typically sampled at 22,050Hz for deep learning, even though this does lose information over 10,000Hz, we save memory and computation time this way (in the same way a human does not take in every instance of a continuous analogue waveform). I made some models that take input as raw mel spectrogram, and others raw waveform.

We can mimic the perception of sound by humans, and can for example apply STFT (approximation of time-frequency analysis performed by the ear [4]), which allows decomposing of signal into its individual frequencies and amplitudes, in order to condense this information and obtain information that can be analysed along the frequency domain (instead of just raw-waveforms and its time-domain). This condenses how much data we have (due to stft), so it is then cheaper to process (plus it provides filters (like embedding) that is similar to how humans filter sound). But there is a catch. When analyse along time domain we will have fine temporal information but high uncertainty of what frequency is exactly at this point in time. And for frequency domain analysis we have fine frequency resolution, but high uncertainty when it occurred. This means we can not get arbitrarily high frequency and temporal resolution [5]. A small window size gives high time resolution, and low frequency resolution; a large window size gives vice versa. Information is lost from not having a overlap of windows. For example this [6] paper demonstrated that best measurements were frame size of 512 and hop size (for all frame sizes) was best at 50%. There were other options: MFCC, raw spectrogram, etc. But I chose to use raw Mel Spectrogram as it is a way of embedding/tokenizing the data (similar to what you would do for NLP with a dictionary of words), which then gives us filter channels (i.e. the deep learning model is given already some useful channels to extract features.

[1] I have made the decision to only develop and run these algorithms locally on my own GPU, and not on Google Collab as I believe it is a waste of resources (not worth carbon footprint).

[2] This will be especially important to realize, particularly for CNN's

[3] Audio data is not static data that can be observed in parallel to identify different features/aspects. It is just a sequence of air pressures (like encoding of information), and we extract/interpret meaning from this information over time

[4] https://ccrma.stanford.edu/ jos/sasp/Short_Time_Fourier_Transform.html

[5] This is called the "uncertainty principle". The more certainty you want in one variable means more distribution/uncertainty in the other.

[6] Short Time Fourier Transform based music genre classification

Which reduces training time).

CNN's enable invariance to the affine transformations, so that the model can recognize patterns that are shifted, warped, moved, etc. When we stack CNN's, we can get better accuracy. The deeper layers should extract more abstract features (this is a paper on explaining features found in sound [3]. This paper is why I chose 5 convolution layers) (due to the accumulated receptive fields of previous shallower layers exposing the higher layers to more of the image, but with loss of resolution due to convolutions and pooling (down-sampling)); the shallower layers will learn low level features like edges. If we believe our data has lower granularity, then we can have larger filter sizes for convolution layers (11x11, 9x9, etc.); otherwise if we think our data is actually very detailed with high granularity then we need to use small filter sizes so our layer looks at smaller chunks of data (For example 3x3. To compare finer details between these smaller chunks).

As spectrogram is 2D, we would think that using 2D convolutions would give good or better prediction results for the model, but this is not the case. Time-frames and mel frequency bands/frequency (the domains) are different units. Moving something in time domain will keep its meaning, but moving it in frequency domain will change its meaning. So 2D CNN for audio don't seem able to take full advantage of 2D convolutions (but we see we can make them work anyway).

RNN Persist past information, so that current information can be compared to previous information (i.e. a sequence). For LSTM's we have choice between stateful [7] or stateless [8]. We only need stateless. We also must choose to process either: (1) only the last output state (i.e. process the sentiment of the series), or process (2) the intermediate outcomes and process them with for example CNN's. I chose option 2, even though the sentiment did give some what impressive result and trick me at first into thinking I was reshaping the output properly as input for the FC's or GAP's.

Related work

A good paper to start with is this introduction to audio signal processing with deep learning [21]. Out of what paths it described, I chose to use CNN architectures and RCNN architectures. I chose to use raw waveform for some models, and raw mel spectogram for others. Another option is to convert music mel spectogram to image, and then do transfer learning using a existing image deep learning network (Imagenet, densenet, etc) [10].

CNN

From looking at papers: [3], [11], [18], [2], [1] for raw spectogram processing deep learning, and [6],[15] for raw waveform deep learning. We also see image classification CNN networks like VGG_16 and AlexNet which, have also been used (usually as base marks) for audio classification deep learning papers. The lesson I took from these papers was 5 layers would work well for raw spectogram input, but for raw waveform we would need a deeper CNN, and definitely narrow convolutions with small pooling sizes.

We see typical CNN architecture patterns for both input data; [conv -> BN -> ReLu -> pooling] [9] blocks stacked on each other (ranging from 5, to 18 layers), with a one or two fully connected layers at the end. The convolution does linear transformation, and the relu is chosen as it introduces non-linearity to the decision function without distorting the receptive fields of convolutions (negligible amount) and is needed to learn non-linear filters. Pooling is to reduce spatial size (and contributes to local translation invariance), while also providing non linearity in the case of max pooling (extract sharp features).

[7] Reset cell memory every epoch. This means if I batch/group a collection of sequences (batched sequences in order), then connections/state will be passed (and manipulated) sequence to sequence, and could then learn of relationship long term over the group of sequences.

[8] Resets state between sequences (i.e. sequences are independent of eachother).

[9] https://cs231n.github.io/convolutional-networks/#layerpat

RCNN

From looking at papers: [18], [4] and [19], we see a very obvios pattern. This pattern is explored in [22], which is:
([CONV->ReLU->pool]*b) -> (LSTM*a) -> ([FC-> relu]*c)->output
Which is the standard pattern discussed in [22] (as a general RCNN architecture). I tried the opposite pattern (reversing convolutions with lstms), but got far worse results. It makes sense that the above pattern is best because then we can apply for example 1d temporal convolutions (using the mel spectrogram as filters for example), and this will get in a non literal sense "snapshots of time", that can than be reshaped into sequences of features/filters; this means the convolution layers extract usefull features and the lstm looks at these in respect to time (i.e. it uses the patterns found by convolutions). But, the other way around means we are looking for patterns in time using lstm, and then CNN is trying to find patterns in the patterns that were found in time, which does not seem as beneficial as the original pattern).
Something I noticed was how they used GRU's instead of LSTMs for RCNN for audio signal processing, but I noticed this to late. I found out the reason for this is because GRUs are better when sequences are short. The sequences I am using are very short (we are using 3s samples), and so GRU is better as will get roughly same accuracy with fewer parameters and shorter training time (according to GRU paper [5])).

Other papers that influenced design

Following the batch normalization paper [14] which advices using batch normalization after convolution layers before non linear activation (i.e. ReLu). This allows us to effect the higher order statistics, but mean and variance remain the same. Also following the dropout paper [24] which advices using dropout after activation, we arrive at a combined advised pattern:
->CONV/FC->BatchNorm->ReLu(or other activation)->Dropout->CONV/FC->

But, when I applied dropout to convolutions performance decreased (I applied dropout at small amount of 0.1 after maxpooling); which is inline with the original paper [16], [8]. Dropout does not just effect the layer it is applied to, but all the layers bellow; this is why the dropout paper does mention that when applying dropout to convolution layers within a CNN it is more about introducing noise, then about "overfitting". By applying dropout to lower layers, this will introduce noise to higher layers. This is exactly what [20] and [16] advice doing, but state it seems to be case dependant. Batch normalization applies regularization already, making dropout after less effective (while still introducing longer training times). It is for this reason I stopped applying dropout in convolutions for most of my models, as they were not deep enough and with enough parameters at start of convolution layers (I think this is why), but I did successfully apply it to my deeper models that took input of raw waveform.

From reading such papers as: [17] , [12] and most clearly [23], I decided to use "all convolutions network" techniques (GAP, bottleneck, and strided convolutions to replace max-pooling) to address my overfitting issues. We use pooling to downsample, so that deeper layers can view more and more of the image with the combined receptive field (translation invariance to the internal representation). Were as with strided convolutions we don't downsample in same way, but actually just reduce the overlap of receptive fields in order to reduce dimensions (instead of downsampling). The specific reason we use max-pooling is because it is non linear and extracts the most activated neurons, and tends to lead to extracting the sharp features (and give worse localization, but better translation invariance); were as average pooling is linear and keeps more data (although it smooths it out) and we get averaged features, that actually mite not be important or really exist (because we are taking everything in to account. At least with max-pooling, the value extracted is known to actually exist in the features. You can not say that for whatever value average pooling gives you). Average pooling does give better localization, but because music is very varying it makes intuitive sense to me that max is kind of our only option here if we are to use traditional pooling because music is so varying in what it will activate that averages will probably be to inaccurate; so max-pooling probably suits better (because of a lot of variance in the data). But, this seems very naive when you think about how music is actually filtered by humans; we will get the 'sharp' features which usually gives us patterns when we have enough parameters(or it memorizes), but there has to be a better way. The most important/distinguishable features are not found by what is loudest (which is what max would do); and so strided convolutions make more sense to use, as then we can in a way learn from what we throw away (and make reducing dimensions something we can learn).

FC layers tend to suffer a lot from overfitting (they don't generalize well), and applying dropout in FC layers means tuning hyper parameters, which works, and is naive as it just drops at random. By replacing FC with GAP layers we can apply structural regularization (rather than randomly applying regularization across the data). By replacing max-pooling with convolutions, and FC layers with GAP layers, I achieved a all convolution network CNN. After looking at resnet I saw that they use GAP and FC layer (in that order), but I did not find this gave me good results, and so I replace both FC layers with GAP layers. Also something to note is how we use average (instead of max) pooling at the end of GAP layers (output is logit in our case: which is the 'raw output'). By applying a 1x1 convolution stack (as we did as our GAP layers), the output channels (which is the same size as number of categories) are each a confidence map for that category; so using average makes sense because it stabilizes the values and averages the activity of the categories confidence map(average confidence for the category), were as max pooling would just throw out all the data other than the max value (which would just lead to large values, which is leads to just more extreme guesses. It would be like letting the loudest voice to be heard only for a vote).

Experimental Setup

The GTZAN dataset consists of 100 30 second song clips per genre for 10 genres. Due to it being a small dataset, I augmented the dataset by splitting clip into 10 equal parts, and made sure each segment had the same size (because some song segments were a tiny bit longer or shorter then 30 seconds). I then resampled at 22050Hz as the original 44.1KHZ was too high for us computationally. I used the raw waveform for some models (similar to sampleCNN), and the raw mel spectogram (calculated using STFT at frame sizes of 1024, 512 and 256). This increased the dataset 10x, which improved results.

Firstly I got better results using frame size of 1024 for my cnn 1D models, compared to the paper [7] which found 512 the best size. I used 1024 frame size for my input data (for raw mel spectrogram), but for LSTM and CNN 1D I used size of 512 because most other papers and models did this.

I used cross entropy loss because this is a multi-classification problem, and this is the same as negative log likelihood loss and LogSoftMax (which is computationally faster and more stable(we are going to get output that is not vastly different in scale than input) than softmax, and faster). Also, seeing as music is in log scale for humans anyway, it makes most sense to to use logsoftmax (or maybe I am making a connection were nothing is there. But my experiments arrived at me getting better results with logsoftmax).

I chose ADAM as my optimizer because of how much faster than SGD it was out of box. If I was to do this again, I would like to use different optimizers. I used learning rate of 0.005 (which seems to be adviced starting point), and 100 epochs.

I experimented and restructured my architectures for 1D/2D CNN ,and RCNN, models from using FC layers, to going "all convolution network". At the end of refactoring a CNN-fc architecture to a ACN-CNN, I did some experimenting with trying to address overfitting with other techniques such as changing to different activation functions. Specifically I looked at leaky relu, and adding drop out at front of the CNN networks so as to add noise (by have 0.1-0.2 dropout. But I found 0.1 was the best). What I found was that if I applied 0.1 dropout at every layer of the network, I would get far worse results (both training and validation accuracy), which makes sense because dropout drops some features between layers during training. Dropout in simple way is just making a random chance certain features in a layer of a network won't get through to the next layer (you will turn them off), and this is what we do when we apply dropout at 50% at end of network in the FC layers (were we can have a larger input to dropout on), so only some features get to contribute to the final output/prediction (and so regularize this way). But when we apply dropout near the front of the network, I think it makes more sense to look at it as introducing noise to the network, but that the noise trickles down and propogates down the network [10].

[10] I wanted to look more into this

I chose ReLu because it was tried and true as something basic to start with. I tried PReLu and Leaky Relu, but did not really find any better results, and got worse results if I applied all over.

For initialization of weights for dense layer (linear) I used Xavier Initialization (bounded random uniform distribution [13]). I used Xavier initialization for initializing linear layer (for relu) [9]. This was because I was getting exploding and vanishing gradient when I initialized originally to 0. I got far better results when I did this.

Results

CNN 1D raw mel spectogram:

Model	training accuracy	validation accuracy	test accuracy
Alex Net	0.985	0.852	0.834
VGG 16	0.821	0.709	0.709
1D CNN (with fc layers without dropout)	0.975	0.762	0.765
1D CNN (with fc layers+dropout)	0.969	0.831	0.832
1D CNN (with GAP)	0.996	0.923	0.941
'ACN CNN 1D' (with fc (without dropout) and convs instead of pooling)	0.990	0.811	0.839
final ACN_CNN1D_GAP	0.998	0.923	0.910

CNN 1D raw waveform:

Model	training accuracy	validation accuracy	test accuracy
sampleCNNVarient (with FC)	0.971	0.724	0.684
sampleCNNVarient (with GAP)	0.999	0.753	0.766
sampleCNNVarient (with GAP and convs replacing afew maxpooling layers)	0.999	0.685	0.673

CNN 2D Raw mel spectogram: CNN 1D raw waveform:

Model	training accuracy	validation accuracy	test accuracy
2D CNN base	0.910	0.585	0.594
2D ACN CNN	0.999	0.850	0.850

RCNN:

Model	training accuracy	validation accuracy	test accuracy
Base LSTM	0.779	0.594	0.615
BASIC LSTM 1D-CNN (mel spectogram)	0.959	0.809	0.771
RCNN 1D (mel spectogram. FC)	0.995	0.832	0.839
RCNN 1D (mel spectogram. FC with convs instead of pools)	0.947	0.797	0.808
RCNN 1D (mel spectogram. GAP)	0.995	0.900	0.904
ACN RCNN 1D (mel spectogram. GAP with convs instead of pools)	0.994	0.871	0.90
RCNN 2D (mel spectogram)	0.409	0.391	0.410
LSTM 1D CNN (raw waveform with FC)	0.507	0.483	0.471
LSTM 1D CNN (raw waveform with GAP)	0.601	0.563	0.571

[11]

The best results came from 'ACN CNN 1D'. Although the 'ACN RCNN 1D' seemed to get close results. Can see that 2D CNN for mel spectogram struggles as expected, even though it takes significantly longer.

[11] You can see the architectures after running jupyter notebooks "Demonstration"s. Then you can view them with tensorboard.

Conclusion and future work

0.1 Things to note:

1. Convolutions with even sized kernels should not be used. This notably resulted in terrible results. Use odd sized convolutions, as that way the convolutions are symmetric (if the convolutions were even sized, then would get aliasing error).
2. Typically small convolutions are 3x3 or 5x5, and large convolutions are 9x9 or 11x11 (7x7 depends). We want small convolutions in this case, as the data is very granular. Although we can find suggestingly good features with wider convolutions (11x11) as Alex net showed, my experimenting found that smaller convolutions (3x3) worked best for my architectures ("better to go deeper than wider"). For raw waveform though (as this data is even more granular), we absolutely only found good features if the convolution sizes were very small (i.e. 3x3).
3. initialize weights and bias's properly (with xavier, He, etc.)
4. Applying padding for strided convolutions that (that replaced pooling) was necessary. This is because it preserves the boarders (from being washed away).
5. GAP seems to show far better results over all. It improved each models performance (some by significant amounts), and always reduced overfitting as seen in demonstration notebooks and results.
6. Convolutions replacing pooling worked sometimes. Maxpooling is a great way to get the low hanging features that are usually easier to spot from patterns. It makes model more complex (more parameters), with longer training, but as explained in Demonstrate notebooks (.ipynb), it means that these convolutions have to learn from the downsampling that they do and find patterns (were as max pooling is less computationally expensive and it is often easier to find patterns in loudest features). Also CNN are already very prone to overfitting, as they have so many parameters. I found when I went too overboard with pumping the channels up for strided convolutions (from for example 512 to 1024), the performance dropped significantly. Were as when were just pumping from 128 to 254 performance increased significantly (see results). So although I originally increased channels, but had to stop doing that due to obvious overfitting from to many parameters.

When I decided what optimizer to use, I first chose ADAM because I mainly wanted to get it to learn fast so I could learn how to use pytorch. ADAM seemed best choice. But, I want to use the other optimizers more.

GAP seems a lot more sound to me. I like that it reduces the hyperparameter tuning that dropout requires. I would have liked to expanded more with exploring the ideas of ACN and "network in network" [17], but I would have had to make deeper networks. So il be buying hardware this summer to explore deep learning more.

I would also like to in summer expand this exploring with using GANs as I saw in [21].

Also, sample CNN (for raw waveform) uses same sized kernels and stride for the convolutions and pooling. This is similar to having for stft (for calculating raw mel spectrogram) a hop size of 100% , by having no overlap of receptive field for the starting convolutions. I wanted to try and make the samplecnn to mimic a stft hop size of 50%, which would mean the pooling would be half the size of convolutions. The reason I want to do this is because mel spectrogram works very well (as my results showed), and it is supposed to mimic human perception of sound. 50% is known to be the best hop size, and I think similar would be found for a samplecnn with 50% (of convolution size) downsampling size .

References

[1] Hareesh Bahuleyan. Music genre classification using machine learning techniques, 2018.

[2] Keunwoo Choi, George Fazekas, and Mark Sandler. Automatic tagging using deep convolutional neural networks, 2016.

[3] Keunwoo Choi, George Fazekas, and Mark Sandler. Explaining deep convolutional neural networks on music classification. 07 2016.

[4] Keunwoo Choi, György Fazekas, Mark Sandler, and Kyunghyun Cho. Convolutional recurrent neural networks for music classification. pages 2392–2396, 03 2017.

[5] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014.

[6] Wei Dai, Chia Dai, Shuhui Qu, Juncheng Li, and Samarjit Das. Very deep convolutional neural networks for raw waveforms, 2016.

[7] Ahmet Elbir, Hamza Ilhan, Gorkem Serbes, and Nizamettin Aydin. Short time fourier transform based music genre classification. pages 1–4, 04 2018.

[8] Christian Garbin, Xingquan Zhu, and Oge Marques. Dropout vs. batch normalization: an empirical study of their impact to deep learning. *Multimedia Tools and Applications*, 79:1–39, 05 2020.

[9] Xavier Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. *Journal of Machine Learning Research - Proceedings Track*, 9:249–256, 01 2010.

[10] Daniel Grzywczak and Grzegorz Gwardys. Deep image features in music information retrieval. volume 60, pages 187–199, 08 2014.

[11] Yoonchang Han, Jaehun Kim, and Kyogu Lee. Deep convolutional neural networks for predominant instrument recognition in polyphonic music. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, PP, 05 2016.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. 7, 12 2015.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015.

[14] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. 02 2015.

[15] Jongpil Lee, Jiyoung Park, Keunhyoung Kim, and Juhan Nam. Samplecnn: End-to-end deep convolutional neural networks using very small filters for music classification. *Applied Sciences*, 8:150, 01 2018.

[16] Xiang Li, Shuo Chen, Xiaolin Hu, and Jian Yang. Understanding the disharmony between dropout and batch normalization by variance shift, 2018.

[17] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network, 2014.

[18] Zain Nasrullah and Yue Zhao. Music artist classification with convolutional recurrent neural networks. pages 1–8, 07 2019.

[19] Zain Nasrullah and Yue Zhao. Music artist classification with convolutional recurrent neural networks. pages 1–8, 07 2019.

[20] Sungheon Park and Nojun Kwak. Analysis on the dropout effect in convolutional neural networks. pages 189–204, 03 2017.

[21] Hendrik Purwins, Bo Li, Tuomas Virtanen, Jan Schlüter, Shuo-Yiin Chang, and Tara Sainath. Deep learning for audio signal processing. *IEEE Journal of Selected Topics in Signal Processing*, 13(2):206–219, 2019.

[22] Tara Sainath, Oriol Vinyals, Andrew Senior, and Hasim Sak. Convolutional, long short-term memory, fully connected deep neural networks. pages 4580–4584, 04 2015.

[23] Jost Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. 12 2014.

[24] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 06 2014.