Audio Classification using Spectrograms with Transfer Learning

MAYANK GULATI FEDERICO FAVIA
gulati | favia @kth.se

January 14, 2020

Abstract

Audio classification has proven to be useful in various areas including critical application such as speech recognition, crime detection, music production etc. To show proof of concept in this paper, we demonstrate audio classification on musical instruments of *NSynth* audio dataset using Convolution Neural Network (CNN) technique by converting *.wav* audio file to spectrogram images. We exploit on the existing state-of-the-art infrastructures of image classification used in *ImageNet* challenge namely *ResNet18, AlexNet, GoogLeNet* with the help of transfer learning by adding custom layers in the end of these deep neural networks. This frugal engineering strategy leverages on pre-trained weights of deep networks where only few upper layers will be trained while most of the other layers are kept frozen to take advantage from end-to-end machine learning trained model. Lower layers in neural networks refer to general features which are problem independent, while higher layers refer to specific features which are problem dependent. We play with that dichotomy to adjust the weights of the network so as to rightly classify among ten classes of musical instruments.

Keywords **– Audio Classification, NSynth dataset, Audio Recognition, Convolutional Neural Network (CNN), Spectrogram, ImageNet, Transfer Learning, AlexNet, ResNet18, GoogLeNet.**

Contents

1 Introduction

Audio processing and classification are becoming immensely popular day by day for instance in authentication tasks to ensure security and data-privacy. Not the least, it spreads also in the fields of music production and speech recognition. For instance the process of determining a particular speaker, provided expression of speech from a set of different registered speakers is referred as speaker identification. Speaker identification is a sub-process of speaker recognition in which an individual speaker is recognized based on the information collected from in the variety of speech signal [1]. Until now the most common and successful approaches have been statistical models such as Hidden Markov Model (HMMs), but also experiments with CNNs on raw speech data [2] have been tried. Nowadays, CNNs have proven very effective in image classification, improving greatly with the advent of large datasets, and show promise for audios. Therefore, we propose to use Convolution Neural Networks (CNNs) for doing audio classification by using transfer learning on state-of-the-art image classification models such as *ResNet18 [3], AlexNet [4], GoogLeNet [5]* by converting musical instrument audios from *NSynth* [6] dataset to images, namely spectrograms.

1.1 Literature study

Audio event recognition, the human-like capability to identify and relate sounds from different audios, is an emerging problem in machine perception. Typically, one of the classical approaches to audio classification is based on perceptual features clustering of sounds based on timbral similarity [7]. Audio scene classification (ASC) has been approached quite efficiently with deep recurrent neural networks in [3] on LITIS Rouen dataset. Environmental sound classification with convolutional neural networks have also been conducted on spectrograms in [8], showing that convolutional neural networks perform better in recognizing specific classes (e.g. dog bark, air conditioner) while relatively poor for sounds with short-scale temporal structure (e.g. drilling, jackhammer). They wonder if CNNs could be used in ensemble with other less complex models, as they seem to focus on distinct aspects of sound events. In [9] the authors explore the interpret-ability of neural networks in the audio domain for doing classification of English spoken digits dataset by using previously proposed technique of layer-wise relevance propagation (LRP), both by processing directly wave-forms or spectrogram representations of the data.

We observe a semantic analogy with comparable problems to audio detection from spectrograms such as object detection in images, which has recently reaped enormous benefits due to powerful CNNs. Due to aggressive growth witnessed in the areas of image classification, a paradigm shift in computer vision algorithms from hand picking features based on thresholds to end-to-end machine learning approaches to improve accuracy levels has motivated research in other areas of signal processing. The credit of this advancement mainly goes to ImageNet project [10], which is a large visual database designed for use in visual object recognition software research. This project conducts an annual software contest, the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [11], where software programs compete to correctly classify and detect objects and scenes. This evolution, as well the work conducted in [12], in which they outperformed raw features models for Acoustic Event Detection (AED) classification task on the *Audio Set* [13], encouraged us to leverage on these results in the area of audio classification with help of knowledge sharing framework called *transfer learning*.

1.1.1 Convolutional neural networks

CNNs are networks similar to conventional neural networks with trainable weights and biases. Each neuron in the network receives some input used for performing a dot product with sliding window of a filter and it is then followed by a non-linear function. The whole network results in a single score function which is differentiable so it can be updated during back-propagation. Scores calculated based on soft-max function help in classification among the given classes at the last step. Moreover, end-to-end training became a popular trend in recent years: instead of training all parts of a model individually, one can instead train them all at once.

1.1.2 Limitations

Essentially, our technique exploits machine vision in order to apply machine hearing [14], but until recently results have not been as satisfying as applying deep learning to visual images, according to [15]. For instance, axes of spectrograms, time and frequency, do not carry the same meaning, and spectral properties of sounds are non-local, while in images similar neighbouring pixel often can be assumed to belong to the same visual object. Finally, temporal awareness needs to be taken into account when feeding image neural networks with visual spectrograms representing sounds because audio is fundamentally serial and temporally dependent.

1.1.3 Transfer learning approach

The traditional transfer learning approach involves training a base network on its corresponding dataset and then copy its first *n* layers, which are problem independent, to the first *n* layers of a target network meant for deploying on different dataset. The remaining layers of the target network, that are problem dependent, are then randomly initialized and trained keeping in mind the objective for the target task. In [16] authors talk about two strategies where one can choose to do back-propagation the errors from the new task into the base features to fine-tune them in order to adapt for new task, or wish to leave the transferred feature layers frozen, meaning that keeping the weights static for transferred layers while training on the new task. They explain that the choice of whether or not to fine-tune the first *n* layers of the target network is highly influenced by the size of the target dataset and the number of parameters in the first *n* layers. Due to high complexity of base networks (*ResNet18 [3], AlexNet [4], GoogLeNet [5]*), we keep the original layers frozen and only training on newly added custom layers in the end. In accordance to recent studies conducted by [17] [18] [19], collectively highlights that these initial layers of neural networks indeed compute features that are fairly general and can be used as starting point by adding custom layers in the end to solve specific classification problem in our case musical instruments from the spectrograms.

1.2 Research questions, hypotheses

In this research project, we are investigating the use of CNNs and transfer learning to analyse spectrogram representations in order to classify among different classes of instruments (namely bass, brass, flute, guitar, keyboard, mallet, organ, reed, string, synth lead and vocal) contrary to the classical approaches. For training on the spectrograms dataset to improve efficiency we are assuming a deep network can fulfil the task on patterns detection and image classification.

In order to classify using transfer learning, we assume to obtain the right combination of general and specific layers. This essentially means that we expect features that transferred from base task (done on ImageNet dataset [10]) perform well in tandem with a new set of layers for the new task.

2 Method(s)

The project exploits the empirical method since the performance of the system is evaluated and improved with training and testing phases. The main steps of the system creation are:

- Building image dataset of mel spectrograms considering parameters discussed in Section 2.2 out of audio dataset in form of *.wav* file from *NSynth* [6] using python library called as *librosa* [20] to obtain image data (RGB α) of shape $40 \times 251 \times 4$, where last channel α corresponding to transparency factor is ignored as it is considered fully opaque i.e equal to 1.

- Spectrograms (Figure 1) creation. They represent the frequency content in the audio over time as a variation of intensity value in an image. The frequency content of milliseconds chunks is stringed together as coloured vertical bars.

- Network Architecture. We rely on a deep neural network, adapting state-of-the-art infrastructures (Section 2.3) for recognizing critical parts and classify 10 different musical instruments based on trained weights after training.

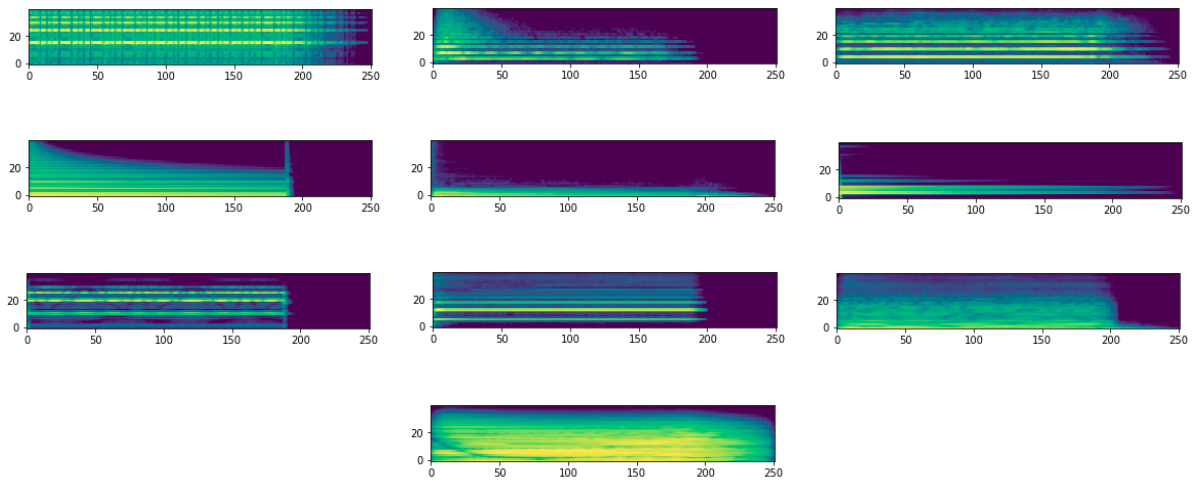

Figure 1: Frequencies on a Mel frequency scale (x-axis) vs equally spaced time (y-axis) for bass, brass, flute, guitar, keyboard, mallet, organ, reed, string and vocal respectively.

2.1 NSynth Dataset

The *NSynth* dataset [6] is a large-scale and high-quality dataset of 305,979 annotated musical notes, each with with a unique pitch, timbre and envelope.

It has been made available by Google Inc. under a Creative Commons Attribution 4.0 International (CC BY 4.0 license). The dataset is available in two different kind of formats: *TFRecord* and JSON files containing non-audio features alongside 16-bit PCM *.wav* audio files. We used this second format of *.wav* files to generate spectrograms. The training set, containing 289,205 samples and the validation set with 12,678 examples (instruments do not overlap with the training set) from 10 different classes were used. The classes are 'bass', 'brass', 'flute', 'guitar', 'keyboard', 'mallet', 'organ', 'reed', 'string' and 'vocal'.

2.2 Framework

Only audio files in *.wav* format, with a single channel are taken into consideration for our research, with the corresponding ground truth labels. Every audio file also has an associated sample rate, which is the number of samples per second of audio.

The amplitude representation over the samples gives a sense of how loud or quiet a change in the air pressure is at any point in time, but for our purpose, a frequency representation is needed. A very common approach to this problem is to keep small overlapping chunks of the signal, and then apply on them Fast Fourier Transform (FFT), which is an efficient computation method to approximate the Discrete Fourier Transform (DFT) (Eq. 1) to move from the time domain to the frequency domain.

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{i2\pi}{N} kn} \tag{1}$$

Frequencies in a signal change over time, thus applying the Fourier transform across the entire signal we would lose the frequency contours of the signal over time. To avoid that, we can safely assume that frequencies in a signal are stationary over an infinitesimal period of time. Therefore, by doing a Fourier transform over this short-time frame, we can obtain a good approximation of the frequency contours of the signal by concatenating adjacent frames.

Doing a N-point FFT on each frame means applying a Short-Time Fourier-Transform (STFT), where N is typically 256, 512 or 1024. The result is converted to polar coordinates giving magnitudes and phases of different frequencies. Only magnitudes are critical for our context, and they are converted to decibel units due to our hearing perception which is based on a logarithmic scale. The power spectrum (periodogram) (2) is computed using the following equation:

$$P = \frac{|FFT(x_i)|^2}{N} \tag{2}$$

where, x_i is the i^{th} frame of signal x. To better model how humans perceive frequencies, also on a logarithmic scale, a melspectrogram representation is more accurate. Humans hear the same distance of frequencies from 50 Hz to 100 Hz as between 400 Hz and 800 Hz. Therefore the frequency bins are converted into the *mel* scale, named by Stevens, Volkmann, and Newman in 1937. According to the popular formulation of Fant (1968) (3), the relation between Hertz (f) and Mel (m) can be obtained using the following equation:

$$m = \frac{1000}{log2} log(1 + \frac{f}{1000}) \tag{3}$$

The regular spectrogram can be changed to a melspectrogram (Figure 1), letting us define how many frequency bins we want, as well as a minimum and maximum frequency.

In particular the parameters we have chosen, according to previous works, are: 1024 points for the window over the FFT is computed; 256 as hop length, meaning that each short frame overlaps with each other of 256 samples; minimum frequency of 20 Hz, maximum frequency equal to half of the sample rate and 40 bins for the mel transformation. Even though it is possible to classify raw audio waveform data, our approach relies on image classifiers to classify these melspectrograms, with acceptable results on test dataset (for us is the validation dataset).

2.3 Network Architecture

The chosen CNN model is adapted version of *ResNet18 [3], AlexNet [4], GoogLeNet [5]*. For instance model summary are shown for customized AlexNet, GoogLeNet, Res18 in Figures 4, 5, 6 used for our purpose where terminal layers are adapted to do classification among 10 musical classes. To better visual the model architectures, its plots can be seen from the link.

3 Results and Analysis

Using different architecture such as adapted *AlexNet [4]* with approximate 62 million trainable parameters, adapted *GoogLeNet [5]* with over 6 million trainable parameters and adapted *ResNet18 [3]* with over 11 million learnable parameters as seen in Figure 4, 5, 6. Analytical evaluations are performed in terms of accuracy using PyTorch [21] native performance metric which calculates commutative correct matches of classes divided by total number of examples conducted over batch size of 64 examples on Microsoft Azure's GPU (Tesla K80) for 500 epochs.

Even though AlexNet model had more parameters to train, it still tries to over-fit on train data and can not perform well on test data. Despite of applying of dropout (where randomly neurons are deactivated to avoid the situation of over-fitting) its performance could not supersede *GoogLeNet [5]* and *ResNet18 [3]* versions. *GoogLeNet [5]* variant significantly improves the performance and also reduces the training time due to pruning of redundant parameters (nearly one tenth of parameters used) compared to AlexNet. Due to devised module called Inception module in *GoogLeNet [5]*, it approximates a sparse CNN with a normal dense construction because only a small number of neurons are effectively participating in deep CNN network.

Another salient point about the module is that it uses convolutions of different sizes to capture details at varied scales (5×5, 3×3, 1×1). As per common strategy by increasing the depth should increase the

accuracy of the network, as long as over-fitting is taken care of. But this leads to the problem of vanishing gradient as during back propagation update of weights becomes less effective with increased depth which essentially causes the earlier layers to learn almost nothing. Residual networks counters this problem by constructing ensembles of many short networks together as considered in *ResNet18 [3]*. Therefore, our Residual network outshines other architectures for our musical instrument classification task using transfer learning as depicted in Table 1.

To better visualize and quantify our results we generated heat map where higher color wavelength like red shows highly correct matches and confusion matrix where diagonal elements show correct classified labels, above diagonal elements are showing false negatives and below diagonal elements are showing false positive elements, in Figures 2 and 3 respectively.

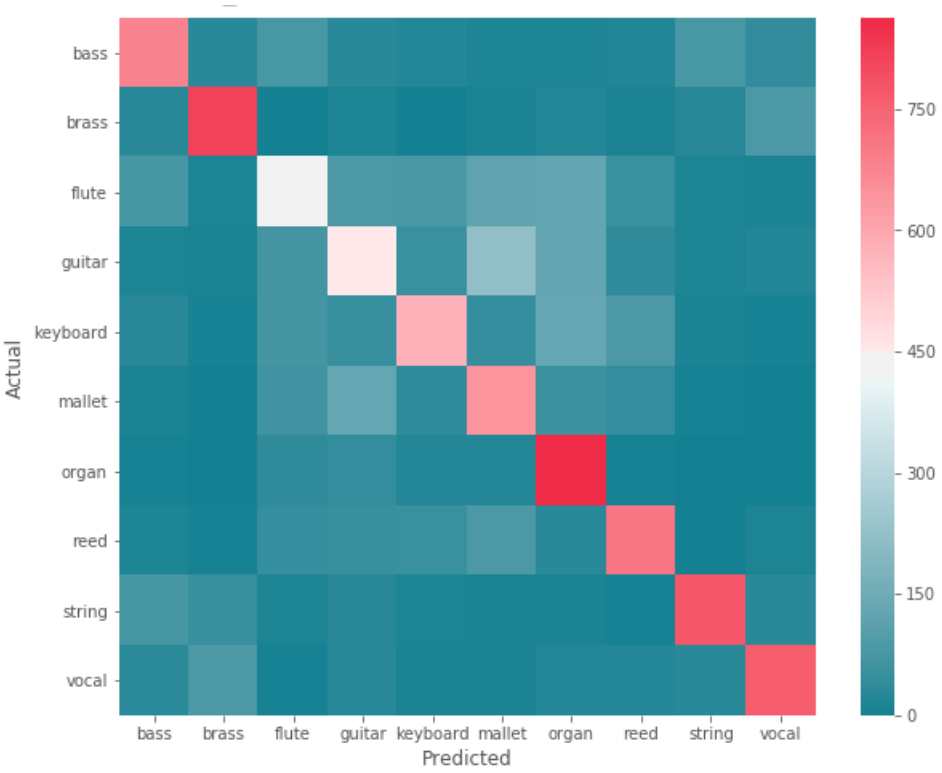

Figure 2: Heat Map showing accuracy in ResNet18

Sr. No.	Model architecture	Accuracy	No. of Parameters
1	AlexNet	0.723	61.8M
2	GoogLeNet	0.762	6.13M
3	ResNet18	0.814	11.24M

Table 1: Comparison among different neural network architecture used under transfer learning

Predicted / Actual	bass	brass	flute	guitar	keyboard	mallet	organ	reed	string	vocal	All
bass	807	19	42	20	9	8	8	15	47	25	1000
brass	20	886	0	12	0	5	13	4	10	50	1000
flute	40	5	675	52	48	67	77	26	8	2	1000
guitar	6	6	38	665	32	126	81	24	10	12	1000
keyboard	15	3	46	25	751	26	76	49	6	3	1000
mallet	3	2	33	73	26	796	34	26	5	2	1000
organ	5	2	18	27	14	14	918	2	0	0	1000
reed	8	4	28	37	34	52	16	814	0	7	1000
string	49	33	9	15	12	7	2	2	850	21	1000
vocal	20	49	3	19	5	4	13	17	14	856	1000
All	973	1009	892	945	931	1105	1238	979	950	978	10000

Figure 3: Confusion matrix showing accuracy in ResNet18

4 Discussion

In summary, we proposed an approach using CNNs and transfer learning for audio classification. The results seen in Table 1 along with analysis conducted in Section 3 affirm that our idea of audio classification using visual spectrogram data has great potential. Despite having one drawback of our approach, the time consuming process of transforming all the audio files into spectrograms, its use on speech data can be explored with help of our approach of deep learning models focusing on transfer learning as it provides interesting footsteps on speaker recognition tasks. Indeed, special emphasis can be given to diphthongs (also referred as a gliding vowel, a combination of two adjacent vowel sounds within the same syllable in a word) detection for effective speaker's classification and recognition. To do so, the speech partitions of Google AudioSet [13] could be investigated.

4.1 Future Work

As future improvements, in order to tackle lack of training visual spectrogram data related to various audio projects, we could use some data augmentation technique to increase the dataset. The regular image transforms such as rotation, flipping, cropping, are not meaningful for spectrograms. The best method would be better to transform raw audio files in the time domain, in order to augment them, and then convert them to spectrograms before being used for classification. Possible transformations can be pitch shifting, time stretching, or simply taking random segments of the audio clips.

An additional improvement could be to exploit PyTorch's frequency transforms method on the GPU allowing to generate spectrograms during training in a considerably faster way. We are also interested in extending our idea further by extracting more information about the musical data for instance detection of musical notes from any instruments which can be really beneficial in the applications of music production.

Finally, our satisfactory results with a single instrument provides promising insights for applications containing multiple audio events detection, for instance in a complete equalized environment, in which more than one instrument could be recognized considering their probabilities.

References

[1] S. Subha and P. Kannan, "Speaker identification techniques – a survey," *International Journal of Advanced and Innovative Research (2278-7844)*, vol. 4 Issue 10, 2015. [Online]. Available: https://www.academia.edu/18551460/Speaker_identification-A_survey

[2] V. Passricha and R. K. Aggarwal, "Convolutional neural networks for raw speech recognition," in *From Natural to Artificial Intelligence*, R. Lopez-Ruiz, Ed. Rijeka: IntechOpen, 2018, ch. 2. [Online]. Available: https://doi.org/10.5772/intechopen.80026

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12. USA: Curran Associates Inc., 2012, pp. 1097–1105. [Online]. Available: http://dl.acm.org/citation.cfm?id=2999134.2999257

[5] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *CoRR*, vol. abs/1409.4842, 2014. [Online]. Available: http://arxiv.org/abs/1409.4842

[6] J. Engel, C. Resnick, A. Roberts, S. Dieleman, D. Eck, K. Simonyan, and M. Norouzi, "Neural audio synthesis of musical notes with wavenet autoencoders," 2017.

[7] P. Herrera-Boyer, G. Peeters, and S. Dubnov, "Automatic classification of musical instrument sounds," *Journal of New Music Research*, vol. 32, no. 1, pp. 3–21, 2003. doi: 10.1076/jnmr.32.1.3.16798. [Online]. Available: https://www.tandfonline.com/doi/abs/10.1076/jnmr.32.1.3.16798

[8] K. J. Piczak, "Environmental sound classification with convolutional neural networks," *2015 IEEE 25th International Workshop on Machine Learning for Signal Processing (MLSP)*, pp. 1–6, 2015.

[9] S. Becker, M. Ackermann, S. Lapuschkin, K.-R. Müller, and W. Samek, "Interpreting and explaining deep neural networks for classification of audio signals," *ArXiv*, vol. abs/1807.03418, 2018.

[10] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[11] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015. doi: 10.1007/s11263-015-0816-y

[12] S. Hershey, S. Chaudhuri, D. P. W. Ellis, J. F. Gemmeke, A. Jansen, R. C. Moore, M. Plakal, D. Platt, R. A. Saurous, B. Seybold, M. Slaney, R. J. Weiss, and K. W. Wilson, "CNN architectures for large-scale audio classification," *CoRR*, vol. abs/1609.09430, 2016. [Online]. Available: http://arxiv.org/abs/1609.09430

[13] J. F. Gemmeke, D. P. W. Ellis, D. Freedman, A. Jansen, W. Lawrence, R. C. Moore, M. Plakal, and M. Ritter, "Audio set: An ontology and human-labeled dataset for audio events," in *Proc. IEEE ICASSP 2017*, New Orleans, LA, 2017.

[14] J. B. Allen, "How do humans process and recognize speech?" *IEEE Trans. on Speech and Audio Proc.*, vol. 2, no. 4, pp. 567–577, Oct. 1994.

[15] L. Wyse, "Audio spectrogram representations for processing with convolutional neural networks," *CoRR*, vol. abs/1706.09559, 2017. [Online]. Available: http://arxiv.org/abs/1706.09559

[16] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 3320–3328. [Online]. Available: http://papers.nips.cc/paper/5347-how-transferable-are-features-in-deep-neural-networks.pdf

[17] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, "Decaf: A deep convolutional activation feature for generic visual recognition," *CoRR*, vol. abs/1310.1531, 2013. [Online]. Available: http://arxiv.org/abs/1310.1531

[18] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," *CoRR*, vol. abs/1311.2901, 2013. [Online]. Available: http://arxiv.org/abs/1311.2901

[19] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," 2013.

[20] B. McFee, M. McVicar, S. Balke, V. Lostanlen, C. Thomé, C. Raffel, D. Lee, Kyungyun Lee, O. Nieto, F. Zalkow, D. Ellis, E. Battenberg, R. Yamamoto, J. Moore, Ziyao Wei, R. Bittner, Keunwoo Choi, Nullmightybofo, P. Friesch, Fabian-Robert Stöter, , Thassilo, M. Vollrath, Siddhartha Kumar Golu, Nehz, S. Waloschek, , Seth, R. Naktinis, D. Repetto, C. Hawthorne, and CJ Carr, "librosa/librosa: 0.6.3," 2019. [Online]. Available: https://zenodo.org/record/2564164

[21] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

A Appendix

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 64, 55, 55]          23,296
              ReLU-2           [-1, 64, 55, 55]               0
         MaxPool2d-3           [-1, 64, 27, 27]               0
            Conv2d-4          [-1, 192, 27, 27]         307,392
              ReLU-5          [-1, 192, 27, 27]               0
         MaxPool2d-6          [-1, 192, 13, 13]               0
            Conv2d-7          [-1, 384, 13, 13]         663,936
              ReLU-8          [-1, 384, 13, 13]               0
            Conv2d-9          [-1, 256, 13, 13]         884,992
             ReLU-10          [-1, 256, 13, 13]               0
           Conv2d-11          [-1, 256, 13, 13]         590,080
             ReLU-12          [-1, 256, 13, 13]               0
        MaxPool2d-13            [-1, 256, 6, 6]               0
AdaptiveAvgPool2d-14            [-1, 256, 6, 6]               0
          Dropout-15                 [-1, 9216]               0
           Linear-16                 [-1, 4096]      37,752,832
             ReLU-17                 [-1, 4096]               0
          Dropout-18                 [-1, 4096]               0
           Linear-19                 [-1, 4096]      16,781,312
             ReLU-20                 [-1, 4096]               0
           Linear-21                 [-1, 1024]       4,195,328
             ReLU-22                 [-1, 1024]               0
           Linear-23                  [-1, 512]         524,800
             ReLU-24                  [-1, 512]               0
           Linear-25                  [-1, 128]          65,664
             ReLU-26                  [-1, 128]               0
           Linear-27                   [-1, 10]           1,290
================================================================
Total params: 61,790,922
Trainable params: 61,790,922
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 8.40
Params size (MB): 235.71
Estimated Total Size (MB): 244.68
----------------------------------------------------------------
```

Figure 4: Model architecture for customized AlexNet

```
----------------------------------------------------
        Layer (type)               Output Shape
====================================================
            Conv2d-1         [-1, 64, 112, 112]
       BatchNorm2d-2         [-1, 64, 112, 112]
       BasicConv2d-3         [-1, 64, 112, 112]
         MaxPool2d-4           [-1, 64, 56, 56]
            Conv2d-5           [-1, 64, 56, 56]
       BatchNorm2d-6           [-1, 64, 56, 56]
       BasicConv2d-7           [-1, 64, 56, 56]
            Conv2d-8          [-1, 192, 56, 56]
       BatchNorm2d-9          [-1, 192, 56, 56]
      BasicConv2d-10          [-1, 192, 56, 56]
        MaxPool2d-11          [-1, 192, 28, 28]
           Conv2d-12           [-1, 64, 28, 28]
      BatchNorm2d-13           [-1, 64, 28, 28]
      BasicConv2d-14           [-1, 64, 28, 28]
           Conv2d-15           [-1, 96, 28, 28]
      BatchNorm2d-16           [-1, 96, 28, 28]
      BasicConv2d-17           [-1, 96, 28, 28]
           Conv2d-18          [-1, 128, 28, 28]
      BatchNorm2d-19          [-1, 128, 28, 28]
      BasicConv2d-20          [-1, 128, 28, 28]
           Conv2d-21           [-1, 16, 28, 28]
      BatchNorm2d-22           [-1, 16, 28, 28]
      BasicConv2d-23           [-1, 16, 28, 28]
           Conv2d-24           [-1, 32, 28, 28]
      BatchNorm2d-25           [-1, 32, 28, 28]
      BasicConv2d-26           [-1, 32, 28, 28]
        MaxPool2d-27          [-1, 192, 28, 28]
           Conv2d-28           [-1, 32, 28, 28]
      BatchNorm2d-29           [-1, 32, 28, 28]
      BasicConv2d-30           [-1, 32, 28, 28]
        Inception-31          [-1, 256, 28, 28]
           Conv2d-32          [-1, 128, 28, 28]
      BatchNorm2d-33          [-1, 128, 28, 28]
      BasicConv2d-34          [-1, 128, 28, 28]
           Conv2d-35          [-1, 128, 28, 28]
      BatchNorm2d-36          [-1, 128, 28, 28]
      BasicConv2d-37          [-1, 128, 28, 28]
           Conv2d-38          [-1, 192, 28, 28]
      BatchNorm2d-39          [-1, 192, 28, 28]
      BasicConv2d-40          [-1, 192, 28, 28]
           Conv2d-41           [-1, 32, 28, 28]
      BatchNorm2d-42           [-1, 32, 28, 28]
      BasicConv2d-43           [-1, 32, 28, 28]
           Conv2d-44           [-1, 96, 28, 28]
      BatchNorm2d-45           [-1, 96, 28, 28]
      BasicConv2d-46           [-1, 96, 28, 28]
        MaxPool2d-47          [-1, 256, 28, 28]
           Conv2d-48           [-1, 64, 28, 28]
      BatchNorm2d-49           [-1, 64, 28, 28]
      BasicConv2d-50           [-1, 64, 28, 28]
        Inception-51          [-1, 480, 28, 28]
        MaxPool2d-52          [-1, 480, 14, 14]
           Conv2d-53          [-1, 192, 14, 14]
      BatchNorm2d-54          [-1, 192, 14, 14]
      BasicConv2d-55          [-1, 192, 14, 14]
           Conv2d-56           [-1, 96, 14, 14]
      BatchNorm2d-57           [-1, 96, 14, 14]
      BasicConv2d-58           [-1, 96, 14, 14]
           Conv2d-59          [-1, 208, 14, 14]
      BatchNorm2d-60          [-1, 208, 14, 14]
      BasicConv2d-61          [-1, 208, 14, 14]
           Conv2d-62           [-1, 16, 14, 14]
      BatchNorm2d-63           [-1, 16, 14, 14]
      BasicConv2d-64           [-1, 16, 14, 14]
           Conv2d-65           [-1, 48, 14, 14]
      BatchNorm2d-66           [-1, 48, 14, 14]
      BasicConv2d-67           [-1, 48, 14, 14]
        MaxPool2d-68          [-1, 480, 14, 14]
           Conv2d-69           [-1, 64, 14, 14]
      BatchNorm2d-70           [-1, 64, 14, 14]
      BasicConv2d-71           [-1, 64, 14, 14]
        Inception-72          [-1, 512, 14, 14]
           Conv2d-73          [-1, 160, 14, 14]
      BatchNorm2d-74          [-1, 160, 14, 14]
      BasicConv2d-75          [-1, 160, 14, 14]
           Conv2d-76          [-1, 112, 14, 14]
      BatchNorm2d-77          [-1, 112, 14, 14]
      BasicConv2d-78          [-1, 112, 14, 14]
           Conv2d-79          [-1, 224, 14, 14]
      BatchNorm2d-80          [-1, 224, 14, 14]
```

```
 BasicConv2d-81          [-1, 224, 14, 14]
      Conv2d-82           [-1, 24, 14, 14]
 BatchNorm2d-83           [-1, 24, 14, 14]
 BasicConv2d-84           [-1, 24, 14, 14]
      Conv2d-85           [-1, 64, 14, 14]
 BatchNorm2d-86           [-1, 64, 14, 14]
 BasicConv2d-87           [-1, 64, 14, 14]
   MaxPool2d-88          [-1, 512, 14, 14]
      Conv2d-89           [-1, 64, 14, 14]
 BatchNorm2d-90           [-1, 64, 14, 14]
 BasicConv2d-91           [-1, 64, 14, 14]
   Inception-92          [-1, 512, 14, 14]
      Conv2d-93          [-1, 128, 14, 14]
 BatchNorm2d-94          [-1, 128, 14, 14]
 BasicConv2d-95          [-1, 128, 14, 14]
      Conv2d-96          [-1, 128, 14, 14]
 BatchNorm2d-97          [-1, 128, 14, 14]
 BasicConv2d-98          [-1, 128, 14, 14]
      Conv2d-99          [-1, 256, 14, 14]
BatchNorm2d-100          [-1, 256, 14, 14]
BasicConv2d-101          [-1, 256, 14, 14]
     Conv2d-102           [-1, 24, 14, 14]
BatchNorm2d-103           [-1, 24, 14, 14]
BasicConv2d-104           [-1, 24, 14, 14]
     Conv2d-105           [-1, 64, 14, 14]
BatchNorm2d-106           [-1, 64, 14, 14]
BasicConv2d-107           [-1, 64, 14, 14]
  MaxPool2d-108          [-1, 512, 14, 14]
     Conv2d-109           [-1, 64, 14, 14]
BatchNorm2d-110           [-1, 64, 14, 14]
BasicConv2d-111           [-1, 64, 14, 14]
  Inception-112          [-1, 512, 14, 14]
     Conv2d-113          [-1, 112, 14, 14]
BatchNorm2d-114          [-1, 112, 14, 14]
BasicConv2d-115          [-1, 112, 14, 14]
     Conv2d-116          [-1, 144, 14, 14]
BatchNorm2d-117          [-1, 144, 14, 14]
BasicConv2d-118          [-1, 144, 14, 14]
     Conv2d-119          [-1, 288, 14, 14]
BatchNorm2d-120          [-1, 288, 14, 14]
BasicConv2d-121          [-1, 288, 14, 14]
     Conv2d-122           [-1, 32, 14, 14]
BatchNorm2d-123           [-1, 32, 14, 14]
BasicConv2d-124           [-1, 32, 14, 14]
     Conv2d-125           [-1, 64, 14, 14]
BatchNorm2d-126           [-1, 64, 14, 14]
BasicConv2d-127           [-1, 64, 14, 14]
  MaxPool2d-128          [-1, 512, 14, 14]
     Conv2d-129           [-1, 64, 14, 14]
BatchNorm2d-130           [-1, 64, 14, 14]
BasicConv2d-131           [-1, 64, 14, 14]
  Inception-132          [-1, 528, 14, 14]
     Conv2d-133          [-1, 256, 14, 14]
BatchNorm2d-134          [-1, 256, 14, 14]
BasicConv2d-135          [-1, 256, 14, 14]
     Conv2d-136          [-1, 160, 14, 14]
BatchNorm2d-137          [-1, 160, 14, 14]
BasicConv2d-138          [-1, 160, 14, 14]
     Conv2d-139          [-1, 320, 14, 14]
BatchNorm2d-140          [-1, 320, 14, 14]
BasicConv2d-141          [-1, 320, 14, 14]
     Conv2d-142           [-1, 32, 14, 14]
BatchNorm2d-143           [-1, 32, 14, 14]
BasicConv2d-144           [-1, 32, 14, 14]
     Conv2d-145          [-1, 128, 14, 14]
BatchNorm2d-146          [-1, 128, 14, 14]
BasicConv2d-147          [-1, 128, 14, 14]
  MaxPool2d-148          [-1, 528, 14, 14]
     Conv2d-149          [-1, 128, 14, 14]
BatchNorm2d-150          [-1, 128, 14, 14]
BasicConv2d-151          [-1, 128, 14, 14]
  Inception-152          [-1, 832, 14, 14]
  MaxPool2d-153            [-1, 832, 7, 7]
     Conv2d-154            [-1, 256, 7, 7]
BatchNorm2d-155            [-1, 256, 7, 7]
BasicConv2d-156            [-1, 256, 7, 7]
     Conv2d-157            [-1, 160, 7, 7]
BatchNorm2d-158            [-1, 160, 7, 7]
BasicConv2d-159            [-1, 160, 7, 7]
     Conv2d-160            [-1, 320, 7, 7]
```

```
      BatchNorm2d-161           [-1, 320, 7, 7]
      BasicConv2d-162           [-1, 320, 7, 7]
           Conv2d-163            [-1, 32, 7, 7]
      BatchNorm2d-164            [-1, 32, 7, 7]
      BasicConv2d-165            [-1, 32, 7, 7]
           Conv2d-166           [-1, 128, 7, 7]
      BatchNorm2d-167           [-1, 128, 7, 7]
      BasicConv2d-168           [-1, 128, 7, 7]
        MaxPool2d-169           [-1, 832, 7, 7]
           Conv2d-170           [-1, 128, 7, 7]
      BatchNorm2d-171           [-1, 128, 7, 7]
      BasicConv2d-172           [-1, 128, 7, 7]
        Inception-173           [-1, 832, 7, 7]
           Conv2d-174           [-1, 384, 7, 7]
      BatchNorm2d-175           [-1, 384, 7, 7]
      BasicConv2d-176           [-1, 384, 7, 7]
           Conv2d-177           [-1, 192, 7, 7]
      BatchNorm2d-178           [-1, 192, 7, 7]
      BasicConv2d-179           [-1, 192, 7, 7]
           Conv2d-180           [-1, 384, 7, 7]
      BatchNorm2d-181           [-1, 384, 7, 7]
      BasicConv2d-182           [-1, 384, 7, 7]
           Conv2d-183            [-1, 48, 7, 7]
      BatchNorm2d-184            [-1, 48, 7, 7]
      BasicConv2d-185            [-1, 48, 7, 7]
           Conv2d-186           [-1, 128, 7, 7]
      BatchNorm2d-187           [-1, 128, 7, 7]
      BasicConv2d-188           [-1, 128, 7, 7]
        MaxPool2d-189           [-1, 832, 7, 7]
           Conv2d-190           [-1, 128, 7, 7]
      BatchNorm2d-191           [-1, 128, 7, 7]
      BasicConv2d-192           [-1, 128, 7, 7]
        Inception-193          [-1, 1024, 7, 7]
AdaptiveAvgPool2d-194           [-1, 1024, 1, 1]
          Dropout-195                 [-1, 1024]
           Linear-196                  [-1, 512]
             ReLU-197                  [-1, 512]
           Linear-198                   [-1, 10]
================================================
Total params: 6,129,834
Trainable params: 6,129,834
Non-trainable params: 0
------------------------------------------------
```

Figure 5: Model architecture for customized GoogLeNet

```
----------------------------------------------
        Layer (type)               Output Shape
==============================================
            Conv2d-1         [-1, 64, 112, 112]
       BatchNorm2d-2         [-1, 64, 112, 112]
              ReLU-3         [-1, 64, 112, 112]
         MaxPool2d-4           [-1, 64, 56, 56]
            Conv2d-5           [-1, 64, 56, 56]
       BatchNorm2d-6           [-1, 64, 56, 56]
              ReLU-7           [-1, 64, 56, 56]
            Conv2d-8           [-1, 64, 56, 56]
       BatchNorm2d-9           [-1, 64, 56, 56]
             ReLU-10           [-1, 64, 56, 56]
       BasicBlock-11           [-1, 64, 56, 56]
           Conv2d-12           [-1, 64, 56, 56]
      BatchNorm2d-13           [-1, 64, 56, 56]
             ReLU-14           [-1, 64, 56, 56]
           Conv2d-15           [-1, 64, 56, 56]
      BatchNorm2d-16           [-1, 64, 56, 56]
             ReLU-17           [-1, 64, 56, 56]
       BasicBlock-18           [-1, 64, 56, 56]
           Conv2d-19          [-1, 128, 28, 28]
      BatchNorm2d-20          [-1, 128, 28, 28]
             ReLU-21          [-1, 128, 28, 28]
           Conv2d-22          [-1, 128, 28, 28]
      BatchNorm2d-23          [-1, 128, 28, 28]
           Conv2d-24          [-1, 128, 28, 28]
      BatchNorm2d-25          [-1, 128, 28, 28]
             ReLU-26          [-1, 128, 28, 28]
       BasicBlock-27          [-1, 128, 28, 28]
           Conv2d-28          [-1, 128, 28, 28]
      BatchNorm2d-29          [-1, 128, 28, 28]
             ReLU-30          [-1, 128, 28, 28]
           Conv2d-31          [-1, 128, 28, 28]
      BatchNorm2d-32          [-1, 128, 28, 28]
             ReLU-33          [-1, 128, 28, 28]
       BasicBlock-34          [-1, 128, 28, 28]
           Conv2d-35          [-1, 256, 14, 14]
      BatchNorm2d-36          [-1, 256, 14, 14]
             ReLU-37          [-1, 256, 14, 14]
           Conv2d-38          [-1, 256, 14, 14]
      BatchNorm2d-39          [-1, 256, 14, 14]
           Conv2d-40          [-1, 256, 14, 14]
      BatchNorm2d-41          [-1, 256, 14, 14]
             ReLU-42          [-1, 256, 14, 14]
       BasicBlock-43          [-1, 256, 14, 14]
           Conv2d-44          [-1, 256, 14, 14]
      BatchNorm2d-45          [-1, 256, 14, 14]
             ReLU-46          [-1, 256, 14, 14]
           Conv2d-47          [-1, 256, 14, 14]
      BatchNorm2d-48          [-1, 256, 14, 14]
             ReLU-49          [-1, 256, 14, 14]
       BasicBlock-50          [-1, 256, 14, 14]
           Conv2d-51            [-1, 512, 7, 7]
      BatchNorm2d-52            [-1, 512, 7, 7]
             ReLU-53            [-1, 512, 7, 7]
           Conv2d-54            [-1, 512, 7, 7]
      BatchNorm2d-55            [-1, 512, 7, 7]
           Conv2d-56            [-1, 512, 7, 7]
      BatchNorm2d-57            [-1, 512, 7, 7]
             ReLU-58            [-1, 512, 7, 7]
       BasicBlock-59            [-1, 512, 7, 7]
           Conv2d-60            [-1, 512, 7, 7]
      BatchNorm2d-61            [-1, 512, 7, 7]
             ReLU-62            [-1, 512, 7, 7]
           Conv2d-63            [-1, 512, 7, 7]
      BatchNorm2d-64            [-1, 512, 7, 7]
             ReLU-65            [-1, 512, 7, 7]
       BasicBlock-66            [-1, 512, 7, 7]
AdaptiveAvgPool2d-67            [-1, 512, 1, 1]
           Linear-68                  [-1, 128]
             ReLU-69                  [-1, 128]
           Linear-70                   [-1, 10]
==============================================
Total params: 11,243,466
Trainable params: 11,243,466
Non-trainable params: 0
----------------------------------------------
```

Figure 6: Model architecture for customized ResNet18