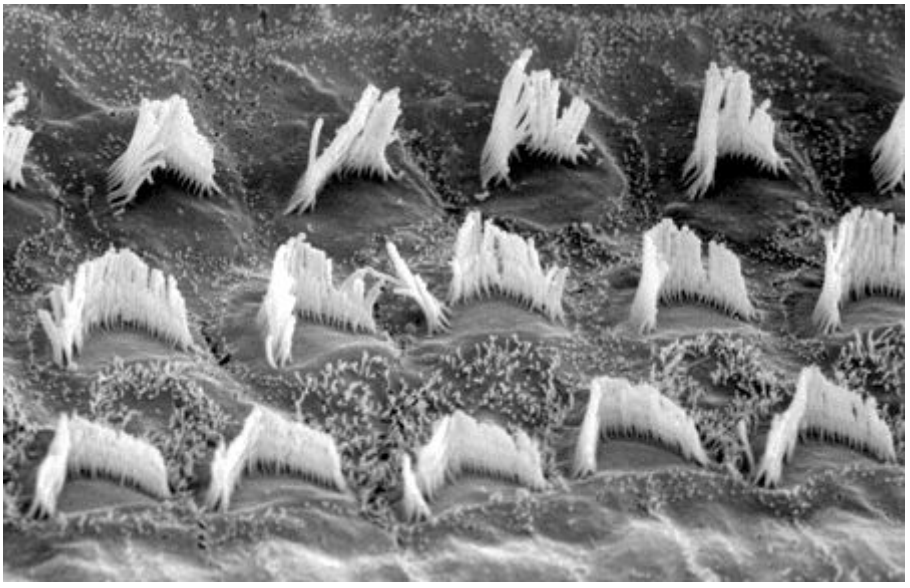
Music Genre Classification Using Convolutional Neural Networks

CS985 Coursework - Vlady Veselinov - Github Repo

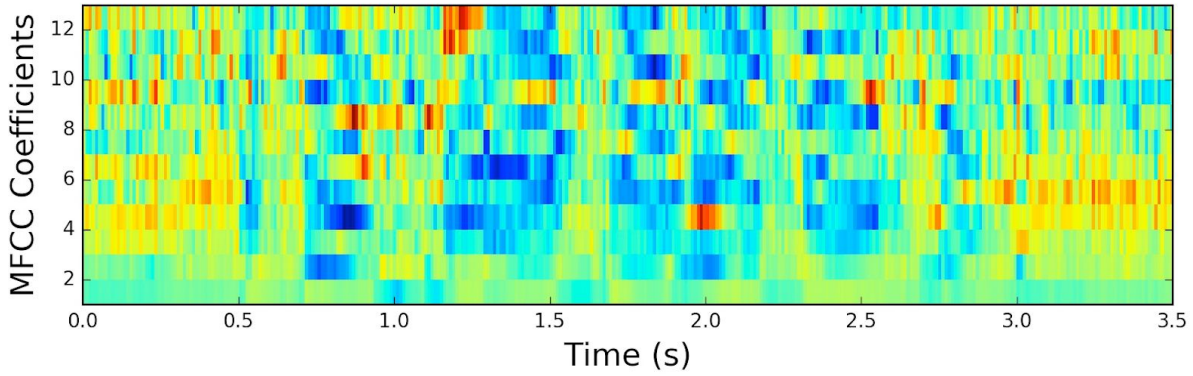
1. Main Ideas and Data

Traditionally, most machine learning effort is concentrated in the sphere of images. Both images and audio can be viewed as signals, therefore much of the techniques that are used for image machine learning can also be used for sounds. One particular topic of concern is the form of the data. Images are usually fed to algorithms as is, without any feature engineering. Audio on the other hand needs a little help. To elaborate on this, if we compare how humans perceive colour, it is evident that the brain doesn't receive raw wave signals to process. The retina contains colour-sensitive cells that send a signal if they're stimulated by a specific colour, i.e. red, green, blue and grey-scale, depending on the type of cell (rod or cone). In a similar fashion, human hearing organs don't send a direct waveform to the brain. A mechanical system converts the vibrations from the eardrum to a fluid inside the inner ear, where in the cochlea, in the organ of Corti, there are many vibration-sensitive hair cells which all resonate at a different frequency. Respectively anywhere from 20 Hz to 20 kHz. These cells can be considered small microphones, with regions dedicated to a specific frequency range.



The point of all this is that image data can be fed straight into neural networks without preprocessing, but raw audio cannot. Specifically because it's in the form of a 1-dimensional wave. Usually sampled at 44.1kHz, that would be 1 323 000 samples per 30 seconds of audio. That's more data compared to a 512 x 512 picture. Plus, the picture would have colour information encoded, audio doesn't have that inherently. For this reason, audio needs to be preprocessed in a way that reflects its "colour", i.e. frequency. The most common way to do this is to use a fast Fourier transform. In the music information retrieval sphere, there

are many useful parameters but this paper will focus on one specific form of data. Mel-frequency cepstral coefficients (MFCC). These are essentially derived in relation to how humans perceive sound with high sensitivity in pitch change in low frequencies and low sensitivity in pitch change in high frequencies. They can be simply described as frequency bins with a coefficient. They are widely used in voice recognition. When plotted, they look like this:



An audio file can be divided into windows of some number of samples to illustrate time horizontally and 13 MFCCs to illustrate simplified frequency content. This is the main idea of the data preprocessing philosophy for this project. The specific dataset used is the Ballroom music dataset, containing roughly 700 music files with a length of 30s with genres like cha cha, rumba, samba, tango, etc. The audio MFCC feature is extracted using the Essentia audio dsp library. In actuality, it can extract more than 200 parameters, but for the sake of scope this project will focus only on utilising MFCCs. The audio files need to be in a folder with subfolders, where the subfolders are regarded as labels used for the machine learning process. Audio features are extracted in bulk and written to JSON files, after which they are distilled into a .csv file with two columns: category and MFCCs. Each row contains roughly 1300 frames with 13 mel-frequency cepstral coefficients each. In data.py, the data is reshaped to be a tensor of the form (nrows, 1300, 13, 1) to suit the input layer of the neural networks.

2. Architecture, Training and Evaluation

The TFLearn library (Tang, 2016) is used in the development of this project because it utilises TensorFlow under the hood and greatly simplifies the development experience by yielding the same intended results at the end. This code in TensorFlow:

```
with tf.name_scope('conv1'):
    W = tf.Variable(tf.random_normal([5, 5, 1, 32]), dtype=tf.float32,
name='Weights')
    b = tf.Variable(tf.random_normal([32]), dtype=tf.float32,
name='biases')
    x = tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')
    x = tf.add_bias(x, b)
    x = tf.nn.relu(x)
```

Looks like this in TFLearn:

```
tflearn.conv_2d(x, 32, 5, activation='relu', name='conv1')
```

This allows for very quick iteration on the model architecture with the added benefit of going low-level if needed.

A simple convolutional neural network is compared to deeper networks to figure out how much network depth affects the model accuracy. This simple network has the following architecture:

```
def simple(len_categories):
	net = input_data(shape=[None, 1300, 13, 1], name='input')
	net = conv_2d(net, 32, 2, activation='relu')
	net = max_pool_2d(net, 2)
	net = conv_2d(net, 64, 2, activation='relu')
	net = max_pool_2d(net, 2)
	net = fully_connected(net, 1024, activation='relu')
	net = dropout(net, 0.8)

	net = tflearn.fully_connected(net, len_categories,
activation='softmax')
	net = regression(net, optimizer='adam', learning_rate=0.001,
loss='categorical_crossentropy', name='targets')

return net
```

It would be unwise to not make use of key research in the area of image recognition, since the audio data can very well represent a picture. In genre recognition literature (Feng, Liu and Yao, 2017) the architecture is predominantly inspired by the popular AlexNet, VGG and GoogleNet image recognition architectures. Some also use a model that combines a convolutional and a recurrent section to make a decision for music genre. This project will do only convolutional for the sake of scope. In the area of music instrument recognition Han, Kim and Lee, 2017 train a model directly on images of spectrograms, without extracting any audio-specific features beforehand. There isn't any other practical way of understanding how architecture affects the accuracy of the model except by doing the training and evaluating each model's performance on a validation set. Each model is trained like this:

```
import tflearn
from models import simple
from data import train_x, train_y, test_x, test_y, categories

simple_model = tflearn.DNN(
	simple(len(categories)),
	tensorboard_verbose=3,
	tensorboard_dir='../logs',
)
```

```
simple_model.fit(
      { 'input': train_x },
      { 'targets': train_y },
      n_epoch=100,
      validation_set=({ 'input': test_x }, { 'targets': test_y }),
      batch_size=64,
      snapshot_step=200,
      show_metric=True,
      run_id='simple'
)

simple_model.save('../trained_models/simple/simple_model')
```

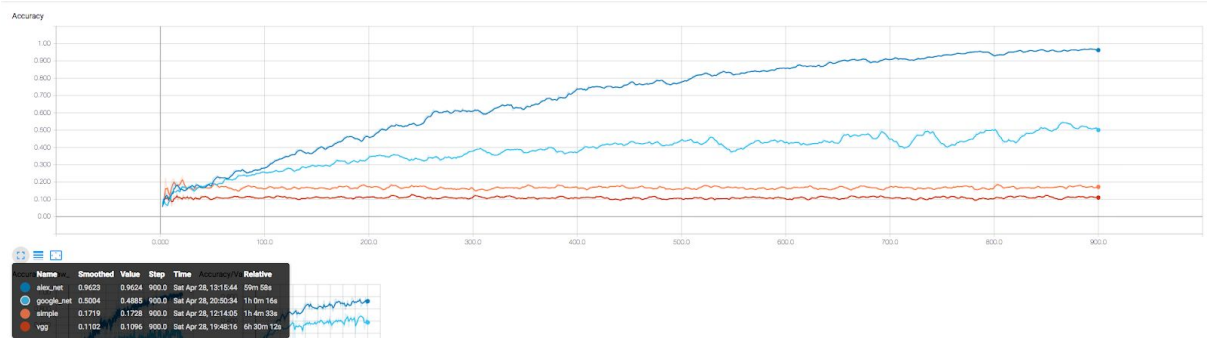
There are separate training files for each model because TensorFlow doesn't like it and throws errors when multiple models are using the same session. For an easy training and evaluation experience a couple of shell scripts are created in the root directory which run all the necessary files.

After generating a train / test data split (80/20). All models are trained for 100 epochs, most taking around an hour, while VGG took 6 hours to train. The resulting trained models are actually quite big (2GB zipped). They can be downloaded here, in order to evaluate them, they need to be unzipped into the root project folder and **evaluate.sh** needs to be ran.

The shell scripts are an elegant way of going around TensorFlow's single session per Python file limitation:

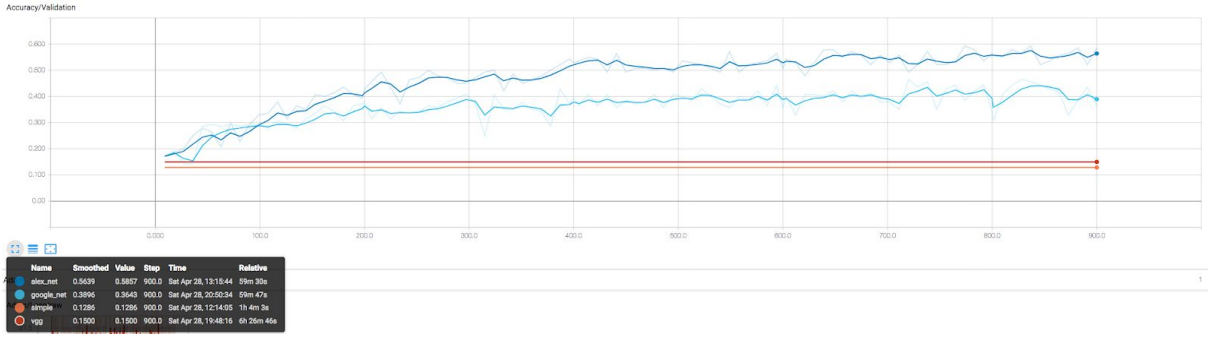
```
#!/bin/sh

cd src
python evaluate_simple.py
python evaluate_alex_net.py
python evaluate_vgg.py
python evaluate_google_net.py
```

After training and evaluation, it becomes clear that VGG (red) and the Simple (orange) model pale in comparison to AlexNet (dark blue) and GoogleNet (light blue):



In the training set, AlexNet can surpass 95% accuracy where GoogleNet is around 50%. Simple is at a low 17% and VGG is even lower at 11%.

Looking at the **validation** set however shows different results:



Name	Smoothed	Value	Step	Time	Relative
alex_net	0.5639	0.5857	900.0	Sat Apr 28, 13:15:44	59m 30s
google_net	0.3896	0.3643	900.0	Sat Apr 28, 20:50:34	59m 47s
simple	0.1286	0.1286	900.0	Sat Apr 28, 12:14:05	1h 4m 3s
vgg	0.1500	0.1500	900.0	Sat Apr 28, 19:48:16	6h 26m 46s

High res images are available here.

This is a very good number for AlexNet which shows that it can be somewhat used in practical applications, given that it was trained only on 700 snippets of 30s audio.

3. Conclusion and Future Work

This project proves the viability of architectures like AlexNet for audio classification applications. The Simple model proves to not be deep enough for this particular problem. It is unclear why VGG isn't performing well. There are little examples out there on how to prepare the audio data for the network, so reflecting back on this it might be useful to focus on data gathering and processing methods for future use. Speaking of future possibilities, this system could be expanded to automatically correct its learning rates for optimal training (Smith, 2015). Experimentally correcting learning rate is very, very tedious, given the above training times for the networks. So albeit out of the scope of this project, it would be very useful to implement automatic tuning of this parameter, because it's probably the most important one when training the neural net. It would be interesting to expand on this project and develop a general purpose classificator that can work for any sound category, not just music genres. This idea will probably be undertaken in a dissertation project aiming to develop a system that can detect sounds within sounds, much like how the modern real-time image detection project You Only Look Once predict bounding boxes around objects in images. (Redmon and Farhadi, 2018)

References

Feng, L., Liu, S. and Yao, J. (2017). Music Genre Classification with Paralleling Recurrent Convolutional Neural Network.

Han, Y., Kim, J. and Lee, K. (2017). Deep Convolutional Neural Networks for Predominant Instrument Recognition in Polyphonic Music. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 25(1), pp.208-221.

Redmon, J. and Farhadi, A. (2018). YOLOv3: An Incremental Improvement.

Smith, L. (2015). Cyclical Learning Rates for Training Neural Networks.

Tang, Y. (2016). TF.Learn: TensorFlow's High-level Module for Distributed Machine Learning.