CNN 2D Basic Solution Powered by fast.ai

This kernel explains basic solution that I've used in the last competition and many of top competitors also. It's CNN, even ImageNet pretrained model works fine with audio 2D image like data.

Will show:

- Converting audio to 2D image like array, so that we can simply exploit strong CNN classifier.
- fast.ai to build fast and strong multi-label classifier model. Unlike normal use, we need to train from scratch to comply competition rule. (Though if we use ImageNet pretrained model, it converges super fast...)
- With simple codes.

Update 30-Apr, 2019

- Now fast.ai library ready to use lwlrap as metric: https://colab.research.google.com/drive/1AgPdhSp7ttY18O3fEoHOQKlt_3HJDLi8 (https://colab.research.google.com/drive/1AgPdhSp7ttY18O3fEoHOQKlt_3HJDLi8)
- And TTA! https://github.com/fastai/fastai/blob/master/fastai/vision/tta.py (https://github.com/fastai/fastai/blob/master/fastai/vision/tta.py) --> Oops, it might not be effective for this problem. Now planning to update one more...

Update 28-Apr, 2019

- Removed EasyDict dependency.
- Training steps improved, tuned by running `lr_find()` and `fit_one_cycle()` iteratively.

In [1]:
```
# This Python 3 environment comes with many helpful analytics libraries inst
alled
# It is defined by the kaggle/python docker image: https://github.com/kaggle
/docker-python
# For example, here's several helpful packages to load in

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from pathlib import Path
import matplotlib.pyplot as plt
from tqdm import tqdm_notebook
import IPython
import IPython.display
import PIL

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will l
ist the files in the input directory

import os
print(os.listdir("../input"))

# Any results you write to the current directory are saved as output.
```

```
['test', 'train_noisy.csv', 'train_curated.csv', 'train_curated', 'sample_sub
mission.csv', 'train_noisy']
```

File/folder definitions

- `df` will handle training data.
- `test_df` will handle test data.

In [2]:
```
DATA = Path('../input')
CSV_TRN_CURATED = DATA/'train_curated.csv'
CSV_TRN_NOISY = DATA/'train_noisy.csv'
CSV_SUBMISSION = DATA/'sample_submission.csv'
TRN_CURATED = DATA/'train_curated'
TRN_NOISY = DATA/'train_noisy'
TEST = DATA/'test'

WORK = Path('work')
IMG_TRN_CURATED = WORK/'image/trn_curated'
IMG_TRN_NOISY = WORK/'image/train_noisy'
IMG_TEST = WORK/'image/test'
for folder in [WORK, IMG_TRN_CURATED, IMG_TRN_NOISY, IMG_TEST]:
    Path(folder).mkdir(exist_ok=True, parents=True)

df = pd.read_csv(CSV_TRN_CURATED)
test_df = pd.read_csv(CSV_SUBMISSION)
```

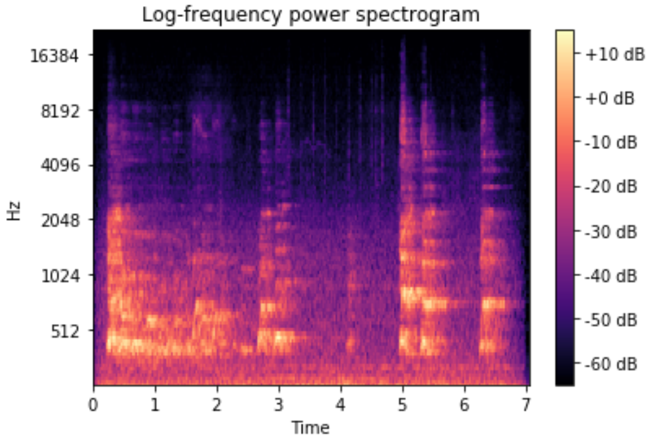
Audio conversion to 2D

Almost copyed from my repository: https://github.com/daisukelab/ml-sound-classifier (https://github.com/daisukelab/ml-sound-classifier)

- Handle sampling rate 44.1kHz as is, no information loss.
- Size of each file will be 128 x L, L is audio seconds x 128; `[128, 256]` if sound is 2s long.
- Convert to Mel-spectrogram, not MFCC. We are handling general sound rather than human voice. https://en.wikipedia.org/wiki/Spectrogram (https://en.wikipedia.org/wiki/Spectrogram)

In [3]:
```
import librosa
import librosa.display

def read_audio(conf, pathname, trim_long_data):
    y, sr = librosa.load(pathname, sr=conf.sampling_rate)
    # trim silence
    if 0 < len(y): # workaround: 0 length causes error
        y, _ = librosa.effects.trim(y) # trim, top_db=default(60)
    # make it unified length to conf.samples
    if len(y) > conf.samples: # long enough
        if trim_long_data:
            y = y[0:0+conf.samples]
    else: # pad blank
        padding = conf.samples - len(y)    # add padding at both ends
        offset = padding // 2
        y = np.pad(y, (offset, conf.samples - len(y) - offset), 'constant')
    return y

def audio_to_melspectrogram(conf, audio):
    spectrogram = librosa.feature.melspectrogram(audio,
                                                 sr=conf.sampling_rate,
                                                 n_mels=conf.n_mels,
                                                 hop_length=conf.hop_length,
                                                 n_fft=conf.n_fft,
                                                 fmin=conf.fmin,
                                                 fmax=conf.fmax)
    spectrogram = librosa.power_to_db(spectrogram)
    spectrogram = spectrogram.astype(np.float32)
    return spectrogram

def show_melspectrogram(conf, mels, title='Log-frequency power spectrogram
'):
    librosa.display.specshow(mels, x_axis='time', y_axis='mel',
                             sr=conf.sampling_rate, hop_length=conf.hop_leng
th,
                            fmin=conf.fmin, fmax=conf.fmax)
    plt.colorbar(format='%+2.0f dB')
    plt.title(title)
    plt.show()

def read_as_melspectrogram(conf, pathname, trim_long_data, debug_display=Fal
se):
    x = read_audio(conf, pathname, trim_long_data)
    mels = audio_to_melspectrogram(conf, x)
    if debug_display:
        IPython.display.display(IPython.display.Audio(x, rate=conf.sampling_
rate))
        show_melspectrogram(conf, mels)
    return mels


class conf:
    # Preprocessing settings
    sampling_rate = 44100
    duration = 2
    hop_length = 347*duration # to make time steps 128
    fmin = 20
    fmax = sampling_rate // 2
    n_mels = 128
    n_fft = n_mels * 20
    samples = sampling_rate * duration

# example
x = read_as_melspectrogram(conf, TRN_CURATED/'0006ae4e.wav', trim_long_data=
False, debug_display=True)
```

13:31:36 / 13:31:36



Making 2D mel-spectrogram data as 2D 3ch images

So that normal CNN image classifier can handle. I wanted to put them into files, but kernel has restriction to keep files less than 500. We need to keep the data on memory.

Of course this has positive effect, training gets faster.

In [4]:
```
def mono_to_color(X, mean=None, std=None, norm_max=None, norm_min=None, eps=
1e-6):
    # Stack X as [X,X,X]
    X = np.stack([X, X, X], axis=-1)

    # Standardize
    mean = mean or X.mean()
    std = std or X.std()
    Xstd = (X - mean) / (std + eps)
    _min, _max = Xstd.min(), Xstd.max()
    norm_max = norm_max or _max
    norm_min = norm_min or _min
    if (_max - _min) > eps:
        # Scale to [0, 255]
        V = Xstd
        V[V < norm_min] = norm_min
        V[V > norm_max] = norm_max
        V = 255 * (V - norm_min) / (norm_max - norm_min)
        V = V.astype(np.uint8)
    else:
        # Just zero
        V = np.zeros_like(Xstd, dtype=np.uint8)
    return V

def convert_wav_to_image(df, source, img_dest):
    X = []
    for i, row in tqdm_notebook(df.iterrows()):
        x = read_as_melspectrogram(conf, source/str(row.fname), trim_long_da
ta=False)
        x_color = mono_to_color(x)
        X.append(x_color)
    return X

X_train = convert_wav_to_image(df, source=TRN_CURATED, img_dest=IMG_TRN_CURA
TED)
X_test = convert_wav_to_image(test_df, source=TEST, img_dest=IMG_TEST)
```

Custom `open_image` for fast.ai library to load data from memory

- Important note: Random cropping 1 sec, this is working like augmentation.

In [5]:
```
from fastai import *
from fastai.vision import *
from fastai.vision.data import *
import random

CUR_X_FILES, CUR_X = list(df.fname.values), X_train

def open_fat2019_image(fn, convert_mode, after_open)->Image:
    # open
    idx = CUR_X_FILES.index(fn.split('/')[-1])
    x = PIL.Image.fromarray(CUR_X[idx])
    # crop
    time_dim, base_dim = x.size
    crop_x = random.randint(0, time_dim - base_dim)
    x = x.crop([crop_x, 0, crop_x+base_dim, base_dim])
    # standardize
    return Image(pil2tensor(x, np.float32).div_(255))

vision.data.open_image = open_fat2019_image
```

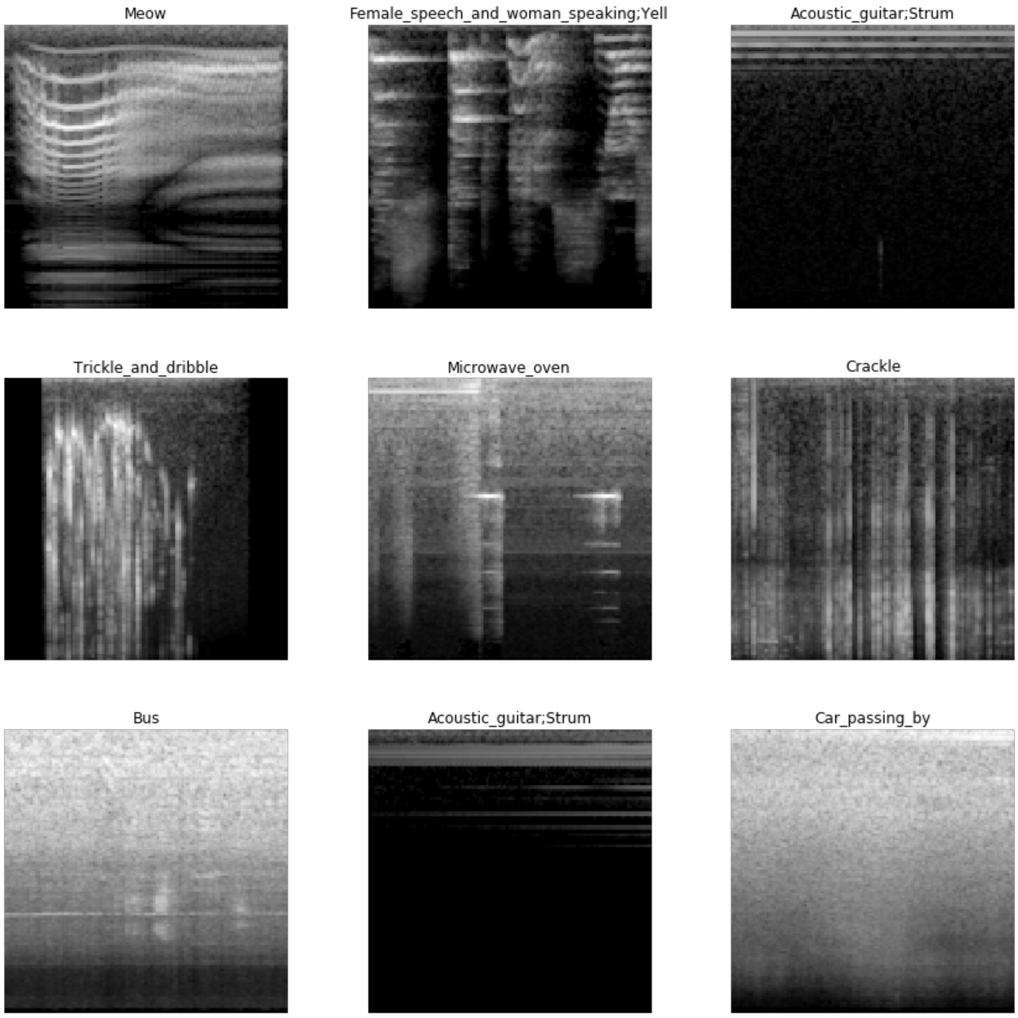
Follow multi-label classification

- Almost following fast.ai course: [https://nbviewer.jupyter.org/github/fastai/course-v3/blob/master/nbs/dl1/lesson3-planet.ipynb (https://nbviewer.jupyter.org/github/fastai/course-v3/blob/master/nbs/dl1/lesson3-planet.ipynb)](https://nbviewer.jupyter.org/github/fastai/course-v3/blob/master/nbs/dl1/lesson3-planet.ipynb)
- But `pretrained=False`
- With lwlrap as metric: [https://colab.research.google.com/drive/1AgPdhSp7ttY18O3fEoHOQKlt_3HJDLi8 (https://colab.research.google.com/drive/1AgPdhSp7ttY18O3fEoHOQKlt_3HJDLi8)](https://colab.research.google.com/drive/1AgPdhSp7ttY18O3fEoHOQKlt_3HJDLi8)

In [6]:
```
# from official code https://colab.research.google.com/drive/1AgPdhSp7ttY180
3fEoHOQKlt_3HJDLi8#scrollTo=cRCaCIb9oguU
def _one_sample_positive_class_precisions(scores, truth):
    """Calculate precisions for each true class for a single sample.

    Args:
      scores: np.array of (num_classes,) giving the individual classifier sc
ores.
      truth: np.array of (num_classes,) bools indicating which classes are t
rue.

    Returns:
      pos_class_indices: np.array of indices of the true classes for this sa
mple.
      pos_class_precisions: np.array of precisions corresponding to each of
those
        classes.
    """
    num_classes = scores.shape[0]
    pos_class_indices = np.flatnonzero(truth > 0)
    # Only calculate precisions if there are some true classes.
    if not len(pos_class_indices):
        return pos_class_indices, np.zeros(0)
    # Retrieval list of classes for this sample.
    retrieved_classes = np.argsort(scores)[::-1]
    # class_rankings[top_scoring_class_index] == 0 etc.
    class_rankings = np.zeros(num_classes, dtype=np.int)
    class_rankings[retrieved_classes] = range(num_classes)
    # Which of these is a true label?
    retrieved_class_true = np.zeros(num_classes, dtype=np.bool)
    retrieved_class_true[class_rankings[pos_class_indices]] = True
    # Num hits for every truncated retrieval list.
    retrieved_cumulative_hits = np.cumsum(retrieved_class_true)
    # Precision of retrieval list truncated at each hit, in order of pos_lab
els.
    precision_at_hits = (
            retrieved_cumulative_hits[class_rankings[pos_class_indices]] /
            (1 + class_rankings[pos_class_indices].astype(np.float)))
    return pos_class_indices, precision_at_hits


def calculate_per_class_lwlrap(truth, scores):
    """Calculate label-weighted label-ranking average precision.

    Arguments:
      truth: np.array of (num_samples, num_classes) giving boolean ground-tr
uth
        of presence of that class in that sample.
      scores: np.array of (num_samples, num_classes) giving the classifier-u
nder-
        test's real-valued score for each class for each sample.

    Returns:
      per_class_lwlrap: np.array of (num_classes,) giving the lwlrap for eac
h
        class.
      weight_per_class: np.array of (num_classes,) giving the prior of each
        class within the truth labels.  Then the overall unbalanced lwlrap i
s
        simply np.sum(per_class_lwlrap * weight_per_class)
    """
    assert truth.shape == scores.shape
    num_samples, num_classes = scores.shape
    # Space to store a distinct precision value for each class on each sampl
e.
    # Only the classes that are true for each sample will be filled in.
    precisions_for_samples_by_classes = np.zeros((num_samples, num_classes))
    for sample_num in range(num_samples):
```

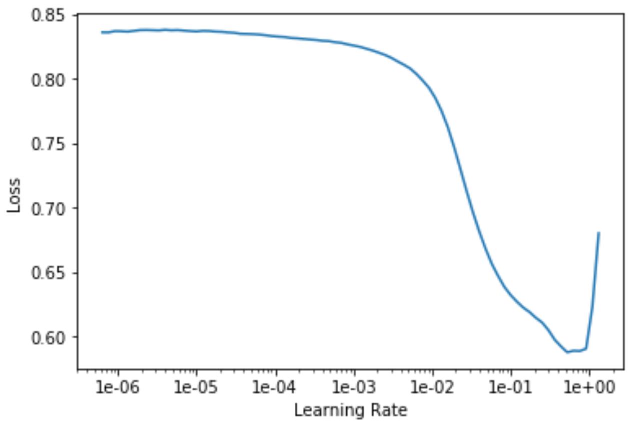
In [7]:
```
tfms = get_transforms(do_flip=True, max_rotate=0, max_lighting=0.1, max_zoo
m=0, max_warp=0.)
src = (ImageList.from_csv(WORK/'image', Path('../../')/CSV_TRN_CURATED, fold
er='trn_curated')
       .split_by_rand_pct(0.2)
       .label_from_df(label_delim=',')
)
data = (src.transform(tfms, size=128)
        .databunch(bs=64).normalize(imagenet_stats)
)
```

In [8]:
```
data.show_batch(3)
```



In [9]:
```
learn = cnn_learner(data, models.resnet18, pretrained=False, metrics=[lwlra
p])
learn.unfreeze()

learn.lr_find(); learn.recorder.plot()
```

```
LR Finder is complete, type {learner_name}.recorder.plot() to see the graph.
```



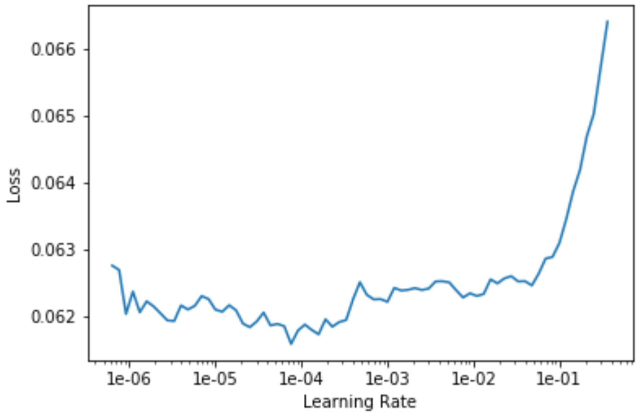
In [10]:
```
learn.fit_one_cycle(5, 1e-1)
learn.fit_one_cycle(10, 1e-2)
```

epoch	train_loss	valid_loss	lwlrap	time
0	0.203928	0.164348	0.091409	00:05
1	0.110289	0.174511	0.086719	00:05
2	0.088189	34.831135	0.070577	00:05
3	0.079456	0.071442	0.148072	00:05
4	0.074203	0.069916	0.178260	00:05

epoch	train_loss	valid_loss	lwlrap	time
0	0.071214	0.069382	0.179116	00:05
1	0.071262	0.069877	0.168510	00:05
2	0.071332	0.069054	0.178944	00:05
3	0.070529	0.068207	0.209876	00:05
4	0.069197	0.068022	0.222181	00:06
5	0.068017	0.065331	0.253232	00:05
6	0.066184	0.063583	0.291280	00:05
7	0.064364	0.061482	0.322215	00:05
8	0.063130	0.060965	0.334768	00:05
9	0.062443	0.060407	0.346225	00:05

In [11]:
```
learn.lr_find(); learn.recorder.plot()
```

```
LR Finder is complete, type {learner_name}.recorder.plot() to see the graph.
```

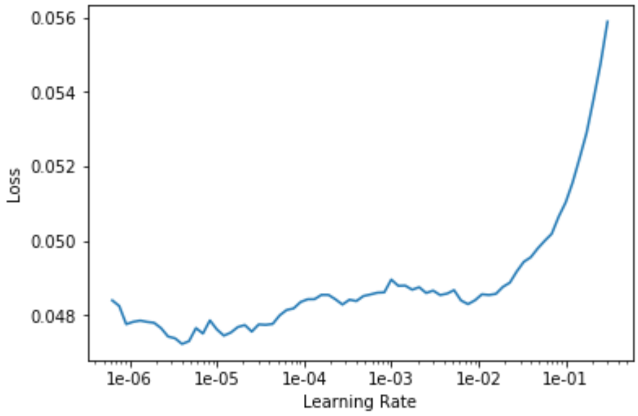


In [12]:
```
learn.fit_one_cycle(20, 3e-3)
```

epoch	train_loss	valid_loss	lwlrap	time
0	0.061830	0.060072	0.353003	00:06
1	0.061821	0.060473	0.343752	00:05
2	0.061613	0.061397	0.325676	00:05
3	0.061907	0.063665	0.270402	00:05
4	0.061584	0.059111	0.362275	00:05
5	0.061148	0.060216	0.346578	00:05
6	0.060348	0.067014	0.233122	00:06
7	0.058780	399.555054	0.188870	00:06
8	0.057434	0.055850	0.421494	00:05
9	0.056827	0.346531	0.298688	00:05
10	0.055272	0.509321	0.358012	00:05
11	0.053970	0.052379	0.459341	00:05
12	0.052903	0.051602	0.464858	00:05
13	0.051684	0.050244	0.482120	00:05
14	0.050607	0.049455	0.490158	00:05
15	0.049708	0.047879	0.516668	00:05
16	0.049585	0.048318	0.513112	00:05
17	0.048978	0.048104	0.524225	00:05
18	0.048675	0.047898	0.515725	00:05
19	0.048509	0.047543	0.523261	00:05

In [13]:
```
learn.lr_find(); learn.recorder.plot()
```

```
LR Finder is complete, type {learner_name}.recorder.plot() to see the graph.
```

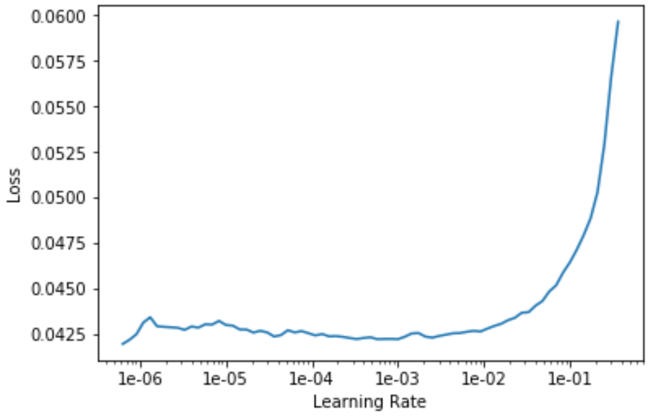


In [14]:
```
learn.fit_one_cycle(20, 1e-3)
```

epoch	train_loss	valid_loss	lwlrap	time
0	0.048523	0.047671	0.524850	00:06
1	0.048265	0.047725	0.521579	00:05
2	0.048441	0.047325	0.523374	00:05
3	0.047853	0.048110	0.517935	00:05
4	0.047490	0.047423	0.529118	00:05
5	0.047575	0.052010	0.466000	00:05
6	0.047299	0.047783	0.522798	00:05
7	0.046861	0.047567	0.524721	00:05
8	0.046786	0.046482	0.546802	00:05
9	0.046325	0.047439	0.530837	00:05
10	0.046137	0.044733	0.564025	00:05
11	0.044961	0.045375	0.554212	00:05
12	0.044631	0.044313	0.569978	00:05
13	0.043908	0.043750	0.571619	00:05
14	0.043461	0.043630	0.573811	00:06
15	0.043340	0.043362	0.577389	00:05
16	0.043057	0.043240	0.576979	00:05
17	0.043017	0.042810	0.585488	00:05
18	0.042827	0.043059	0.589987	00:05
19	0.042646	0.042677	0.594204	00:05

In [15]:
```
learn.lr_find(); learn.recorder.plot()
```

```
LR Finder is complete, type {learner_name}.recorder.plot() to see the graph.
```



In [16]:
```
learn.fit_one_cycle(50, slice(1e-3, 3e-3))
```

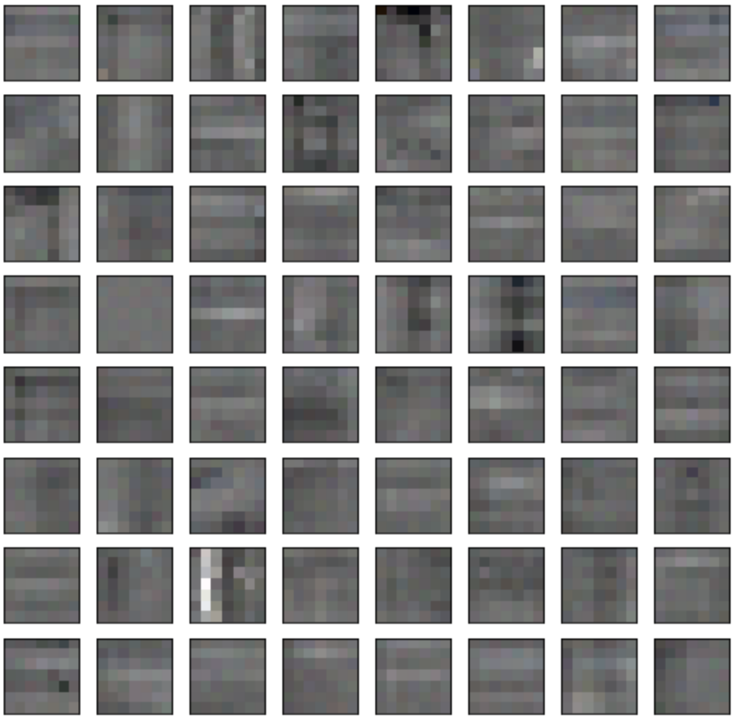
epoch	train_loss	valid_loss	lwlrap	time
0	0.042792	0.042987	0.591302	00:05
1	0.042829	0.043255	0.582411	00:05
2	0.042896	0.042537	0.597984	00:05
3	0.042663	0.042821	0.585062	00:05
4	0.042635	0.043676	0.573729	00:05
5	0.041898	0.042845	0.595862	00:05
6	0.042173	0.043182	0.591107	00:05
7	0.042239	0.043035	0.589884	00:06
8	0.042053	0.042307	0.593668	00:05
9	0.041809	0.042662	0.592296	00:05
10	0.041105	0.045866	0.568954	00:06
11	0.041468	0.043079	0.583152	00:05
12	0.040667	0.041568	0.611382	00:06
13	0.040155	0.042698	0.589495	00:06
14	0.039914	0.045051	0.568335	00:05
15	0.039596	0.041103	0.600563	00:05
16	0.038829	0.102065	0.618892	00:05
17	0.038059	0.039925	0.628607	00:05
18	0.037134	0.041238	0.609029	00:05
19	0.036731	0.041046	0.621627	00:05
20	0.037021	0.041659	0.607099	00:05
21	0.035565	0.040948	0.634019	00:06
22	0.035286	0.039562	0.627891	00:05
23	0.034515	0.038799	0.646282	00:05
24	0.033882	0.039174	0.632907	00:05
25	0.033809	0.037980	0.660209	00:05
26	0.032646	0.038466	0.645996	00:05
27	0.031780	0.042973	0.669017	00:05
28	0.031742	0.036103	0.669407	00:05
29	0.030604	0.036406	0.671344	00:05
30	0.029932	0.036484	0.671596	00:05
31	0.029633	0.036867	0.677426	00:05
32	0.028911	0.037193	0.674700	00:05
33	0.028254	0.038575	0.666470	00:05
34	0.028165	0.035694	0.672885	00:05
35	0.027446	0.035908	0.678962	00:06
36	0.027256	0.036606	0.674998	00:06
37	0.026586	0.034827	0.698813	00:05
38	0.026198	0.035819	0.677323	00:05
39	0.025808	0.035268	0.697643	00:05
40	0.025226	0.035285	0.678577	00:05
41	0.024841	0.035886	0.683629	00:05
42	0.024859	0.034702	0.693509	00:05
43	0.024501	0.034835	0.696768	00:05
44	0.024456	0.034782	0.692983	00:05

In [17]:
```
learn.fit_one_cycle(10, slice(1e-4, 1e-3))
```

epoch	train_loss	valid_loss	lwlrap	time
0	0.023965	0.035153	0.694046	00:06
1	0.024181	0.034172	0.694478	00:05
2	0.024520	0.035063	0.690447	00:05
3	0.024338	0.040201	0.682518	00:05
4	0.024613	0.035896	0.683450	00:05
5	0.024057	0.034727	0.695909	00:05
6	0.023886	0.034688	0.704743	00:05
7	0.022997	0.034314	0.697536	00:05
8	0.023156	0.034111	0.701091	00:05
9	0.023103	0.034318	0.700705	00:05

Let's check how filters are

In [18]:
```
# https://discuss.pytorch.org/t/how-to-visualize-the-actual-convolution-filt
ers-in-cnn/13850
from sklearn.preprocessing import minmax_scale

def visualize_first_layer(learn, save_name=None):
    conv1 = list(learn.model.children())[0][0]
    if isinstance(conv1, torch.nn.modules.container.Sequential):
        conv1 = conv1[0] # for some models, 1 layer inside
    weights = conv1.weight.data.cpu().numpy()
    weights_shape = weights.shape
    weights = minmax_scale(weights.ravel()).reshape(weights_shape)
    fig, axes = plt.subplots(8, 8, figsize=(8,8))
    for i, ax in enumerate(axes.flat):
        ax.imshow(np.rollaxis(weights[i], 0, 3))
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)
    if save_name:
        fig.savefig(str(save_name))

visualize_first_layer(learn)
```



In [19]:
```
learn.save('fat2019_fastai_cnn2d_stage-2')
learn.export()
```

Test prediction and making submission file simple

- Switch to test data.
- Overwrite results to sample submission; simple way to prepare submission file.
- Now using TTA (Test Time Augmentation)!

In [20]:
```
CUR_X_FILES, CUR_X = list(test_df.fname.values), X_test

test = ImageList.from_csv(WORK/'image', Path('../..')/CSV_SUBMISSION, folde
r='test')
learn = load_learner(WORK/'image', test=test)
preds, _ = learn.TTA(ds_type=DatasetType.Test) # <== Simply replacing from l
earn.get_preds()
```

In [21]:
```
test_df[learn.data.classes] = preds
test_df.to_csv('submission.csv', index=False)
test_df.head()
```

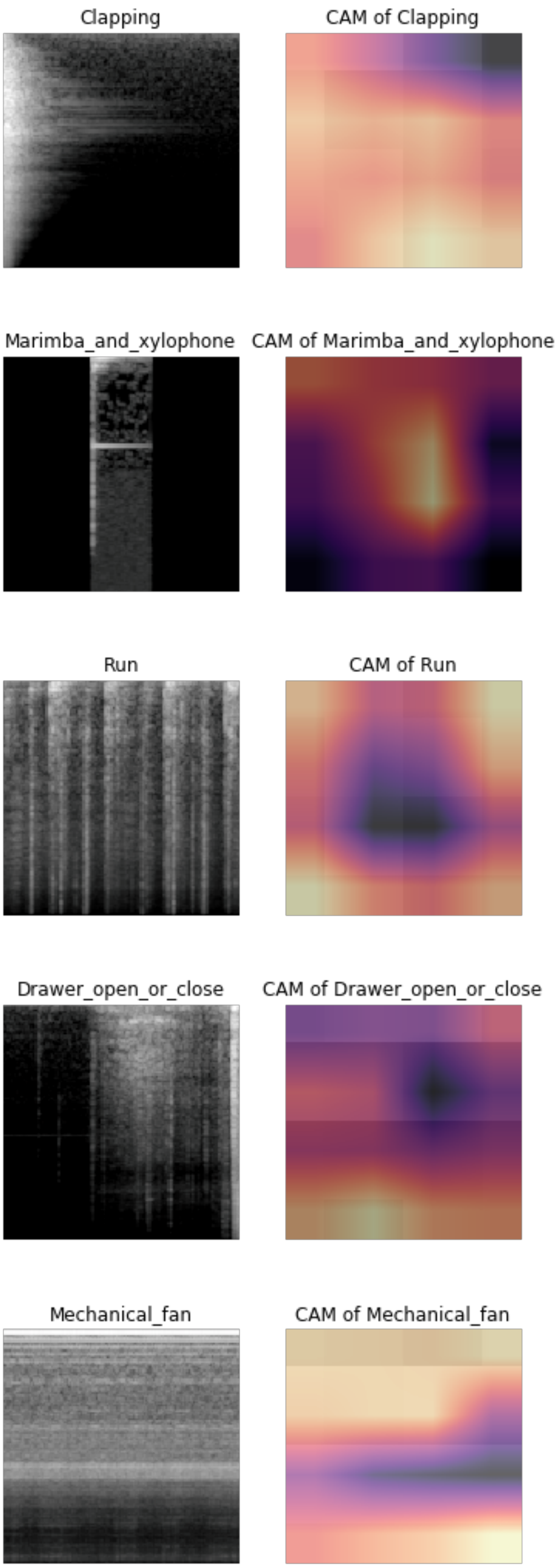
Out[21]:

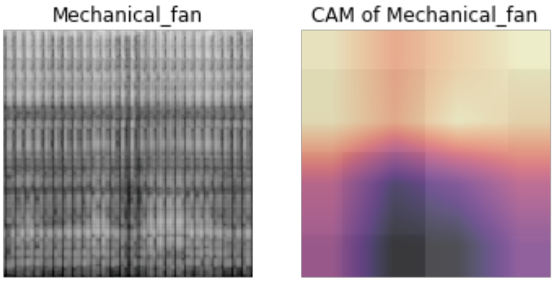
	fname	Accelerating_and_revving_and_vroom	Accordion	Acoustic_guitar	Applause	
0	000ccb97.wav	0.000013	9.970533e-08	3.473719e-08	1.873726e-07	1.3019
1	0012633b.wav	0.069694	3.735964e-05	9.331258e-05	1.868762e-04	2.7503
2	001ed5f1.wav	0.000024	6.408121e-05	4.881012e-07	5.531685e-05	8.6941
3	00294be0.wav	0.000541	1.968583e-06	4.641604e-06	2.855069e-04	3.0243
4	003fde7a.wav	0.000043	2.708495e-05	2.935131e-06	1.166476e-06	6.3599

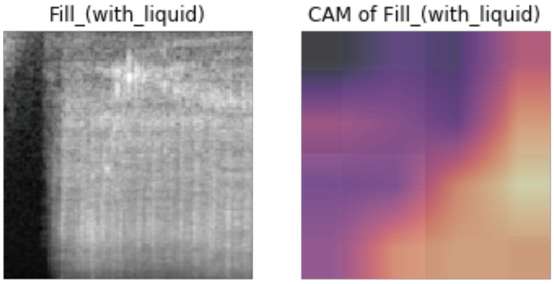
In [22]:
```
CUR_X_FILES, CUR_X = list(df.fname.values), X_train
learn = cnn_learner(data, models.resnet18, pretrained=False, metrics=[lwlra
p])
learn.load('fat2019_fastai_cnn2d_stage-2');
```

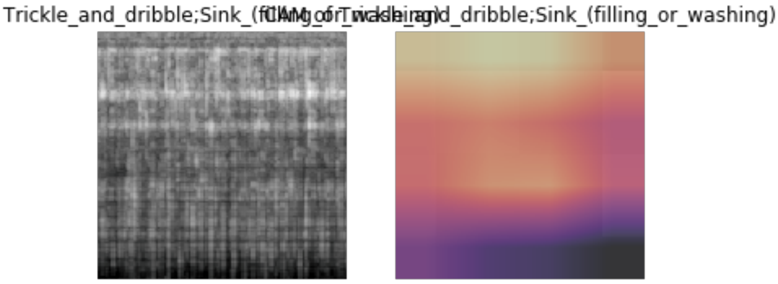
Visualize by CAM

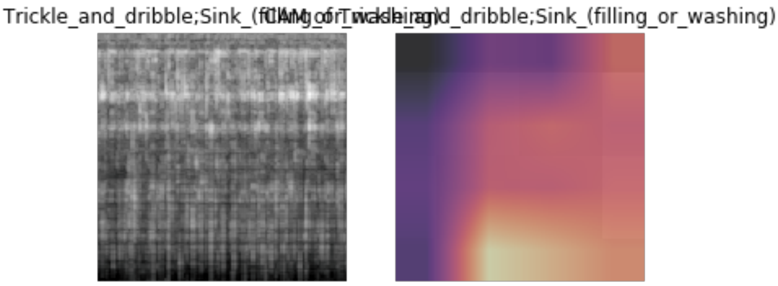
In [23]:
```
# Thanks to https://nbviewer.jupyter.org/github/fastai/course-v3/blob/master
/nbs/dl1/lesson6-pets-more.ipynb
from fastai.callbacks.hooks import *

def visualize_cnn_by_cam(learn, data_index):
    x, _y = learn.data.valid_ds[data_index]
    y = _y.data
    if not isinstance(y, (list, np.ndarray)): # single label -> one hot enco
ding
        y = np.eye(learn.data.valid_ds.c)[y]

    m = learn.model.eval()
    xb,_ = learn.data.one_item(x)
    xb_im = Image(learn.data.denorm(xb)[0])
    xb = xb.cuda()

    def hooked_backward(cat):
        with hook_output(m[0]) as hook_a:
            with hook_output(m[0], grad=True) as hook_g:
                preds = m(xb)
                preds[0,int(cat)].backward()
        return hook_a,hook_g
    def show_heatmap(img, hm, label):
        _,axs = plt.subplots(1, 2)
        axs[0].set_title(label)
        img.show(axs[0])
        axs[1].set_title(f'CAM of {label}')
        img.show(axs[1])
        axs[1].imshow(hm, alpha=0.6, extent=(0,img.shape[0],img.shape[0],0),
                      interpolation='bilinear', cmap='magma');
        plt.show()

    for y_i in np.where(y > 0)[0]:
        hook_a,hook_g = hooked_backward(cat=y_i)
        acts = hook_a.stored[0].cpu()
        grad = hook_g.stored[0][0].cpu()
        grad_chan = grad.mean(1).mean(1)
        mult = (acts*grad_chan[...,None,None]).mean(0)
        show_heatmap(img=xb_im, hm=mult, label=str(learn.data.valid_ds.y[dat
a_index]))

for idx in range(10):
    visualize_cnn_by_cam(learn, idx)
```

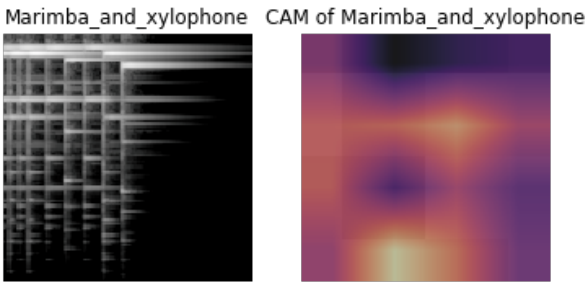

Clapping
CAM of Clapping
Marimba_and_xylophone
CAM of Marimba_and_xylophone
Run
CAM of Run
Drawer_open_or_close
CAM of Drawer_open_or_close
Mechanical_fan
CAM of Mechanical_fan

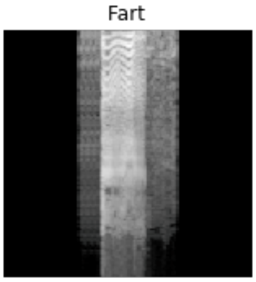

Mechanical_fan
CAM of Mechanical_fan

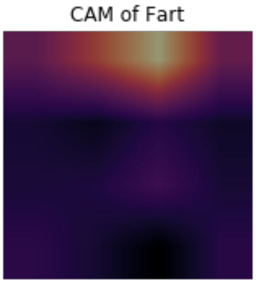

Fill_(with_liquid)
CAM of Fill_(with_liquid)


Trickle_and_dribble;Sink_(filling_or_washing)
CAM of Trickle_and_dribble;Sink_(filling_or_washing)


Trickle_and_dribble;Sink_(filling_or_washing)
CAM of Trickle_and_dribble;Sink_(filling_or_washing)


Marimba_and_xylophone
CAM of Marimba_and_xylophone





In [24]:

```

```

Freesound Audio Tagging 2019

updated May.02

@fizzbuzz's awesome kernel from previous competition would be a great introduction for beginners, including me :)
(https://www.kaggle.com/fizzbuzz/beginner-s-guide-to-audio-data (https://www.kaggle.com/fizzbuzz/beginner-s-guide-to-audio-data))
Here I posted the modified kernel for this competition (though not perfect).

Also some top solutions in previous competition will help us.

- 1st solution : https://storage.googleapis.com/kaggle-forum-message-attachments/365414/9991/Jeong_COCAI_task2.pdf (https://storage.googleapis.com/kaggle-forum-message-attachments/365414/9991/Jeong_COCAI_task2.pdf)
- 4th solution : https://www.kaggle.com/c/freesound-audio-tagging/discussion/62634#latest-367166 (https://www.kaggle.com/c/freesound-audio-tagging/discussion/62634#latest-367166)
- 8th solution : https://www.kaggle.com/c/freesound-audio-tagging/discussion/64262#latest-376395 (https://www.kaggle.com/c/freesound-audio-tagging/discussion/64262#latest-376395)
- 11th solution : http://dcase.community/documents/workshop2018/proceedings/DCASE2018Workshop_Wei_100.pdf (http://dcase.community/documents/workshop2018/proceedings/DCASE2018Workshop_Wei_100.pdf)

- DCASE_2018 proceedings : http://dcase.community/workshop2018/proceedings (http://dcase.community/workshop2018/proceedings)

And more...

Planet Understanding the Amazon from Space was a multi-labeled image classification competition.
https://www.kaggle.com/c/planet-understanding-the-amazon-from-space (https://www.kaggle.com/c/planet-understanding-the-amazon-from-space)

1st place solution had been written in Kaggle blog by @bestfitting.
http://blog.kaggle.com/2017/10/17/planet-understanding-the-amazon-from-space-1st-place-winners-interview/ (http://blog.kaggle.com/2017/10/17/planet-understanding-the-amazon-from-space-1st-place-winners-interview/)
Most interesting part for me is the way to consider co-occurence.
In this solution, Ridge regression was used to do it (please read the above material for more detail).

NOTE :
This notebook used only curated wav files,
and did not consider multi-labeled records in train.
For supplement, I have also posted the kernel to explore multi-label audio data. https://www.kaggle.com/maxwell110/explore-multi-labeled-data (https://www.kaggle.com/maxwell110/explore-multi-labeled-data)

Contents

In [1]:
```
# Change this to True for full dataset and learning
COMPLETE_RUN = False
```

Loading data

In [2]:
```
import numpy as np
np.random.seed(1001)

import os
import shutil
import warnings

import IPython
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from tqdm import tqdm_notebook
from sklearn.model_selection import StratifiedKFold

%matplotlib inline
matplotlib.style.use('ggplot')
warnings.filterwarnings("ignore", category=FutureWarning)
```

In [3]:
```
os.listdir('../input/')
```

Out[3]:
```
['train_curated.csv',
 'train_noisy.csv',
 'test',
 'sample_submission.csv',
 'train_curated',
 'train_noisy']
```

In [4]:
```
train = pd.read_csv("../input/train_curated.csv")
test = pd.read_csv("../input/sample_submission.csv")
```

In [5]:
```
train.sample(10)
```

Out[5]:

	fname	labels
1492	4de932d0.wav	Male_speech_and_man_speaking,Female_speech_and...
4727	f396ec97.wav	Traffic_noise_and_roadway_noise
146	07dd3742.wav	Car_passing_by
3217	a6e6c971.wav	Cutlery_and_silverware
4579	ecb1852a.wav	Male_singing
685	24334f89.wav	Toilet_flush
700	25033c9b.wav	Sigh
1286	43788b3b.wav	Stream
1159	3c65185e.wav	Toilet_flush
1963	6564cce8.wav	Harmonica

In [6]:
```
test.sample(5)
```

Out[6]:

	fname	Accelerating_and_revving_and_vroom	Accordion	Acoustic_guitar	Applause	Bark	Ba
134	07dff283.wav	0	0	0	0	0	
866	342cc3ae.wav	0	0	0	0	0	
180	0a64010f.wav	0	0	0	0	0	
185	0aee7a2a.wav	0	0	0	0	0	
901	3623b13a.wav	0	0	0	0	0	

In [7]:
```
print("Number of train examples=", train.shape[0], "  Number of classes=", l
en(set(train.labels)))
print("Number of test examples=", test.shape[0], "  Number of classes=", len
(set(test.columns[1:])))
```

```
Number of train examples= 4970   Number of classes= 213
Number of test examples= 1120   Number of classes= 80
```

Due to multi-labeld records in train, the number of unique classes is 213 (> 80).

Distribution of Categories

For simplicity, we excluded multi-labeled records in train, so the number of unique label is 74 (< 80). When bulid a valid model, we must consider this.

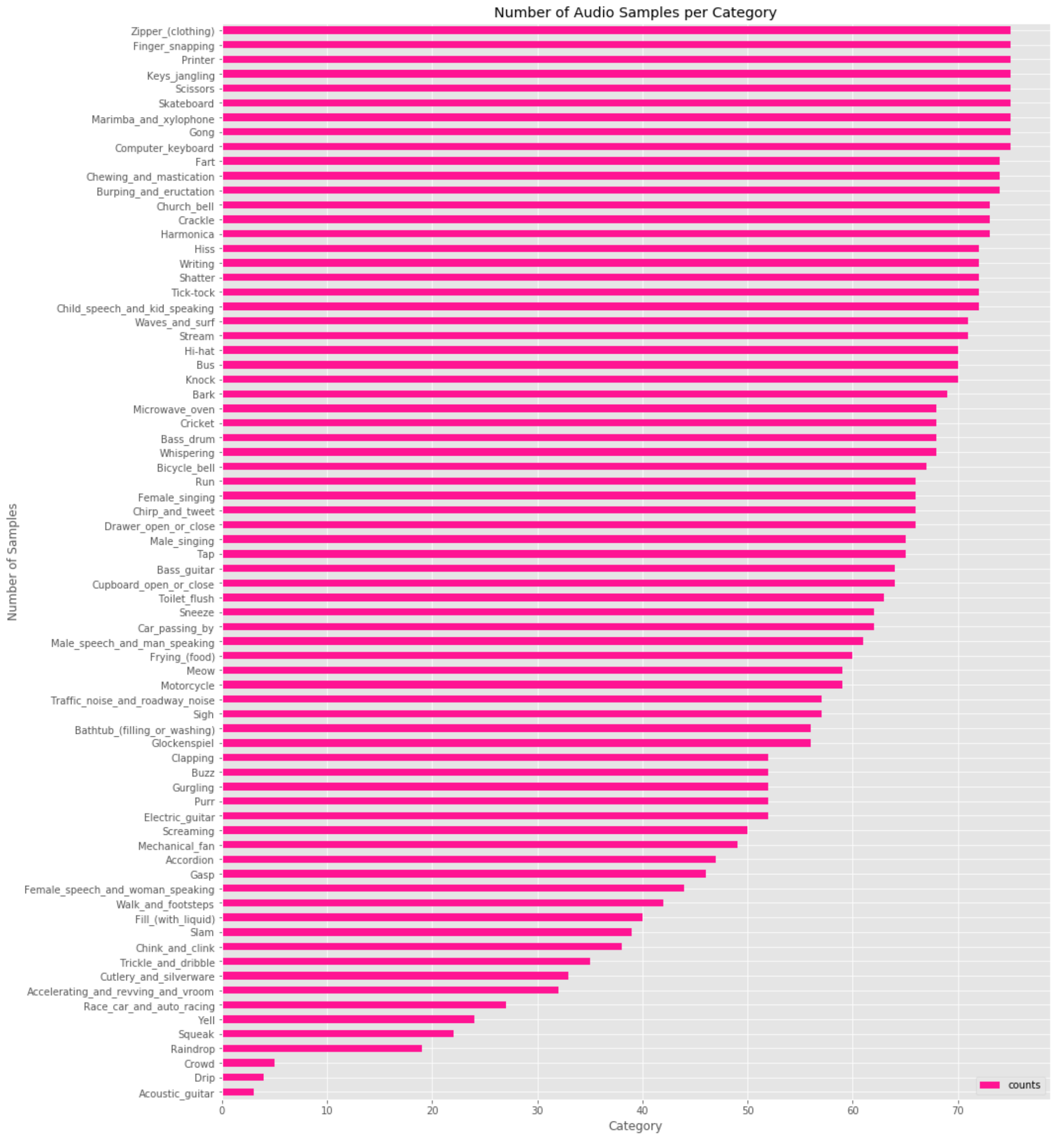
In [8]:
```
train = train[train.labels.isin(test.columns[1:])]
print(len(train))
```

```
4269
```

In [9]:
```
category_group = train.groupby(['labels']).count()
category_group.columns = ['counts']
print(len(category_group))
```

```
74
```

In [10]:
```
plot = category_group.sort_values(ascending=True, by='counts').plot(
    kind='barh',
    title="Number of Audio Samples per Category",
    color='deeppink',
    figsize=(15,20))
plot.set_xlabel("Category")
plot.set_ylabel("Number of Samples");
```

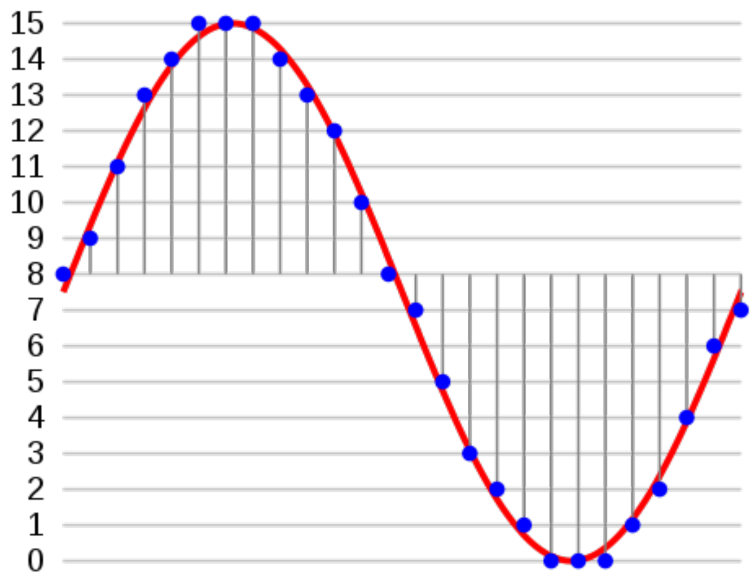


In [11]:
```
print('Minimum samples per category = ', min(train.labels.value_counts()))
print('Maximum samples per category = ', max(train.labels.value_counts()))
```

```
Minimum samples per category =  3
Maximum samples per category =  75
```

Reading Audio Files

The audios are Pulse-code modulated (https://en.wikipedia.org/wiki/Audio_bit_depth) with a bit depth (https://en.wikipedia.org/wiki/Audio_bit_depth) of 16 and a sampling rate (https://en.wikipedia.org/wiki/Sampling_%28signal_processing%29) of 44.1 kHz



- **Bit-depth = 16**: The amplitude of each sample in the audio is one of 2^16 (=65536) possible values.
- **Samplig rate = 44.1 kHz**: Each second in the audio consists of 44100 samples. So, if the duration of the audio file is 3.2 seconds, the audio will consist of 44100*3.2 = 141120 values.

Let's listen to an audio file in our dataset and load it to a numpy array

In [12]:
```
import IPython.display as ipd  # To play sound in the notebook
fname = '../input/train_curated/0019ef41.wav'   # Raindrop
ipd.Audio(fname)
```

Out[12]:

0:00 / 0:02

In [13]:
```
# Using wave library
import wave
wav = wave.open(fname)
print("Sampling (frame) rate = ", wav.getframerate())
print("Total samples (frames) = ", wav.getnframes())
print("Duration = ", wav.getnframes()/wav.getframerate())
```

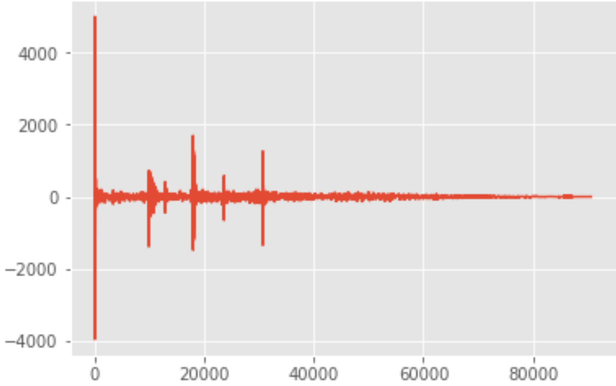
```
Sampling (frame) rate =  44100
Total samples (frames) =  90616
Duration =  2.054784580498866
```

In [14]:
```
# Using scipy
from scipy.io import wavfile
rate, data = wavfile.read(fname)
print("Sampling (frame) rate = ", rate)
print("Total samples (frames) = ", data.shape)
print(data)
```

```
Sampling (frame) rate =  44100
Total samples (frames) =  (90616,)
[ 369 -577  -49 ...    0    0    0]
```

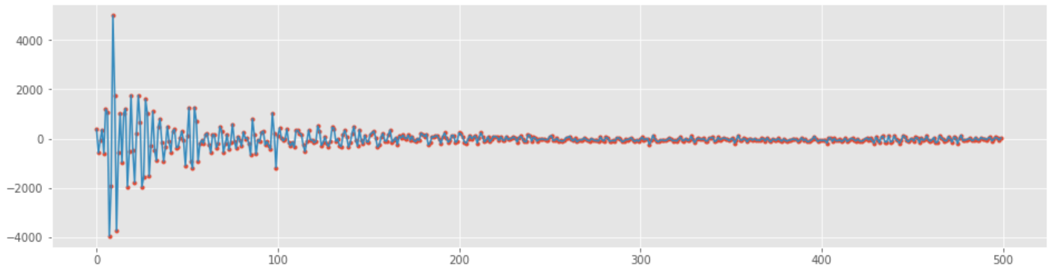
Let's plot the audio frames

In [15]:
```
plt.plot(data, '-', );
```



Let's zoom in on first 1000 frames

In [16]:
```
plt.figure(figsize=(16, 4))
plt.plot(data[:500], '.'); plt.plot(data[:500], '-');
```



Audio Length

We shall now analyze the lengths of the audio files in our dataset

In [17]:
```
train['nframes'] = train['fname'].apply(lambda f: wave.open('../input/train_
curated/' + f).getnframes())
test['nframes'] = test['fname'].apply(lambda f: wave.open('../input/test/' +
f).getnframes())
```

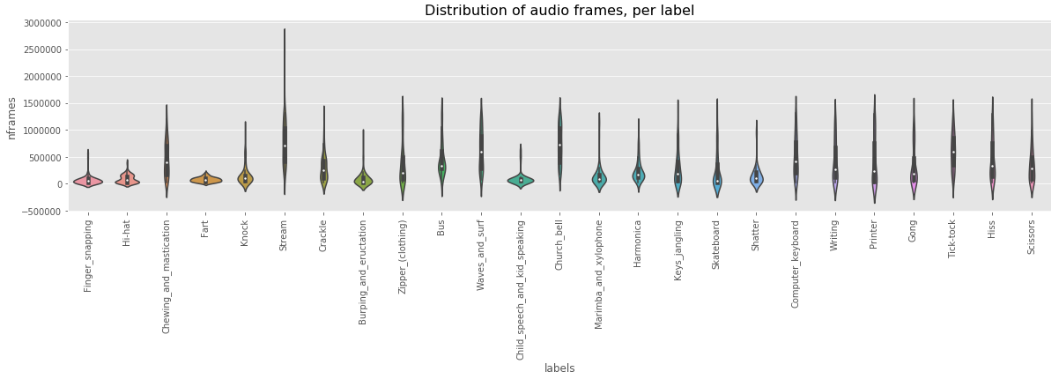
In [18]:
```
train.head()
```

Out[18]:

	fname	labels	nframes
0	0006ae4e.wav	Bark	310456
1	0019ef41.wav	Raindrop	90616
2	001ec0ad.wav	Finger_snapping	66976
3	0026c7cb.wav	Run	1125886
4	0026f116.wav	Finger_snapping	60638

The number of categories is large, so let's check the frame distributions of top 25 categories.

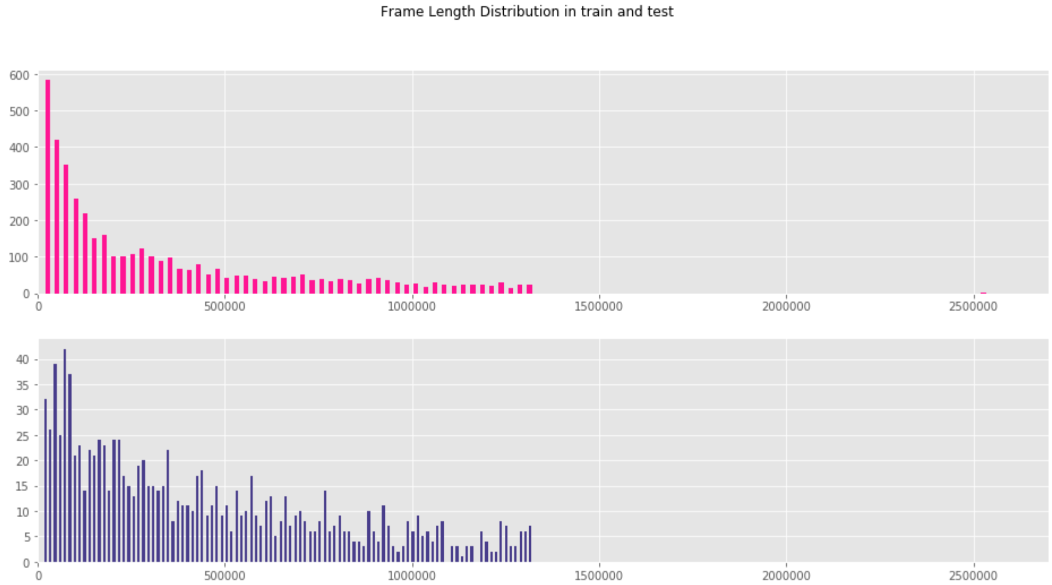
In [19]:
```
idx_sel = category_group.sort_values(ascending=True, by='counts').index[-2
5:]
_, ax = plt.subplots(figsize=(20, 4))
sns.violinplot(ax=ax, x="labels", y="nframes", data=train[(train.labels.isin
(idx_sel).values)])
plt.xticks(rotation=90)
plt.title('Distribution of audio frames, per label', fontsize=16)
plt.show()
```



We observe:
The distribution of audio length across labels is non-uniform and has high variance as the previous competition.

Let's now analyze the frame length distribution in train and test.

In [20]:
```
fig, ax = plt.subplots(2, 1, figsize=(16,8))
train.nframes.hist(bins=100, grid=True, rwidth=0.5, ax=ax[0], color='deeppin
k')
test.nframes.hist(bins=100, grid=True, rwidth=0.5, ax=ax[1], color='darkslat
eblue')
ax[0].set_xlim(0, 2700000)
ax[1].set_xlim(0, 2700000)
plt.suptitle('Frame Length Distribution in train and test', ha='center', fon
tsize='large');
```



We observe:

- Majority of the audio files are short.
- There are an `abnormal` length in the train histogram. Let's analyze them.

In [21]:
```
train.query("nframes > 2500000")
```

Out[21]:

	fname	labels	nframes
2316	77b925c2.wav	Stream	2538889

In [22]:
```
abnormal_length = 2538889

abnormal_fnames = train.loc[train.nframes == abnormal_length, 'fname'].value
s[0]
ipd.Audio( '../input/train_curated/' + abnormal_fnames)
```

Out[22]:

0:00 / 0:58

2. Building a Model using Raw Wave

We will build two models:

1. The first model will take the raw audio (1D array) as input and the primary operation will be Conv1D
2. The second model will take the MFCCs as input. (We will explain MFCC later)

Keras Model using raw wave

Our model has the architecture as follows:

rav

Important: Due to the time limit on Kaggle Kernels, it is not possible to perform 10-fold training of a large model. I have trained the model locally and uploaded its output files as a dataset. If you wish to train the bigger model, change `COMPLETE_RUN = True` at the beginning of the kernel.

Some sssential imports

In [23]:
```
import librosa
import numpy as np
import scipy
from keras import losses, models, optimizers
from keras.activations import relu, softmax
from keras.callbacks import (EarlyStopping, LearningRateScheduler,
                             ModelCheckpoint, TensorBoard, ReduceLROnPlatea
u)
from keras.layers import (Convolution1D, Dense, Dropout, GlobalAveragePoolin
g1D,
                          GlobalMaxPool1D, Input, MaxPool1D, concatenate)
from keras.utils import Sequence, to_categorical
```

```
Using TensorFlow backend.
```

Configuration

The Configuration object stores those learning parameters that are shared between data generators, models, and training functions. Anything that is `global` as far as the training is concerned can become the part of Configuration object.

In [24]:
```
class Config(object):
    def __init__(self,
                 sampling_rate=16000, audio_duration=2,
                 n_classes=len(category_group),
                 use_mfcc=False, n_folds=10, learning_rate=0.0001,
                 max_epochs=50, n_mfcc=20):
        self.sampling_rate = sampling_rate
        self.audio_duration = audio_duration
        self.n_classes = n_classes
        self.use_mfcc = use_mfcc
        self.n_mfcc = n_mfcc
        self.n_folds = n_folds
        self.learning_rate = learning_rate
        self.max_epochs = max_epochs

        self.audio_length = self.sampling_rate * self.audio_duration
        if self.use_mfcc:
            self.dim = (self.n_mfcc, 1 + int(np.floor(self.audio_length/51
2)), 1)
        else:
            self.dim = (self.audio_length, 1)
```

DataGenerator Class

The DataGenerator class inherits from **`keras.utils.Sequence`** . It is useful for preprocessing and feeding the data to a Keras model.

- Once initialized with a batch_size, it computes the number of batches in an epoch. The **`__len__`** method tells Keras how many batches to draw in each epoch.
- The **`__getitem__`** method takes an index (which is the batch number) and returns a batch of the data (both X and y) after calculating the offset. During test time, only `X` is returned.
- If we want to perform some action after each epoch (like shuffle the data, or increase the proportion of augmented data), we can use the **`on_epoch_end`** method.

Note: **`Sequence`** are a safer way to do multiprocessing. This structure guarantees that the network will only train once on each sample per epoch which is not the case with generators.

In [25]:
```
class DataGenerator(Sequence):
    def __init__(self, config, data_dir, list_IDs, labels=None,
                 batch_size=64, preprocessing_fn=lambda x: x):
        self.config = config
        self.data_dir = data_dir
        self.list_IDs = list_IDs
        self.labels = labels
        self.batch_size = batch_size
        self.preprocessing_fn = preprocessing_fn
        self.on_epoch_end()
        self.dim = self.config.dim

    def __len__(self):
        return int(np.ceil(len(self.list_IDs) / self.batch_size))

    def __getitem__(self, index):
        indexes = self.indexes[index*self.batch_size:(index+1)*self.batch_si
ze]
        list_IDs_temp = [self.list_IDs[k] for k in indexes]
        return self.__data_generation(list_IDs_temp)

    def on_epoch_end(self):
        self.indexes = np.arange(len(self.list_IDs))

    def __data_generation(self, list_IDs_temp):
        cur_batch_size = len(list_IDs_temp)
        X = np.empty((cur_batch_size, *self.dim))

        input_length = self.config.audio_length
        for i, ID in enumerate(list_IDs_temp):
            file_path = self.data_dir + ID

            # Read and Resample the audio
            data, _ = librosa.core.load(file_path, sr=self.config.sampling_r
ate,
                                        res_type='kaiser_fast')

            # Random offset / Padding
            if len(data) > input_length:
                max_offset = len(data) - input_length
                offset = np.random.randint(max_offset)
                data = data[offset:(input_length+offset)]
            else:
                if input_length > len(data):
                    max_offset = input_length - len(data)
                    offset = np.random.randint(max_offset)
                else:
                    offset = 0
                data = np.pad(data, (offset, input_length - len(data) - offs
et), "constant")

            # Normalization + Other Preprocessing
            if self.config.use_mfcc:
                data = librosa.feature.mfcc(data, sr=self.config.sampling_ra
te,
                                                   n_mfcc=self.config.n_mfc
c)
                data = np.expand_dims(data, axis=-1)
            else:
                data = self.preprocessing_fn(data)[:, np.newaxis]
            X[i,] = data

        if self.labels is not None:
            y = np.empty(cur_batch_size, dtype=int)
            for i, ID in enumerate(list_IDs_temp):
                y[i] = self.labels[ID]
            return X, to_categorical(y, num_classes=self.config.n_classes)
        else:
```

Normalization

Normalization is a crucial preprocessing step. The simplest method is rescaling the range of features to scale the range in [0, 1].

In [26]:
```
def audio_norm(data):
    max_data = np.max(data)
    min_data = np.min(data)
    data = (data-min_data)/(max_data-min_data+1e-6)
    return data - 0.5
```

- The dummy model is just for debugging purpose.
- Our 1D Conv model is fairly deep and is trained using Adam Optimizer with a learning rate of 0.0001

In [27]:
```
def get_1d_dummy_model(config):

    nclass = config.n_classes
    input_length = config.audio_length

    inp = Input(shape=(input_length,1))
    x = GlobalMaxPool1D()(inp)
    out = Dense(nclass, activation=softmax)(x)

    model = models.Model(inputs=inp, outputs=out)
    opt = optimizers.Adam(config.learning_rate)

    model.compile(optimizer=opt, loss=losses.categorical_crossentropy, metri
cs=['acc'])
    return model

def get_1d_conv_model(config):

    nclass = config.n_classes
    input_length = config.audio_length

    inp = Input(shape=(input_length,1))
    x = Convolution1D(16, 9, activation=relu, padding="valid")(inp)
    x = Convolution1D(16, 9, activation=relu, padding="valid")(x)
    x = MaxPool1D(16)(x)
    x = Dropout(rate=0.1)(x)

    x = Convolution1D(32, 3, activation=relu, padding="valid")(x)
    x = Convolution1D(32, 3, activation=relu, padding="valid")(x)
    x = MaxPool1D(4)(x)
    x = Dropout(rate=0.1)(x)

    x = Convolution1D(32, 3, activation=relu, padding="valid")(x)
    x = Convolution1D(32, 3, activation=relu, padding="valid")(x)
    x = MaxPool1D(4)(x)
    x = Dropout(rate=0.1)(x)

    x = Convolution1D(256, 3, activation=relu, padding="valid")(x)
    x = Convolution1D(256, 3, activation=relu, padding="valid")(x)
    x = GlobalMaxPool1D()(x)
    x = Dropout(rate=0.2)(x)

    x = Dense(64, activation=relu)(x)
    x = Dense(1028, activation=relu)(x)
    out = Dense(nclass, activation=softmax)(x)

    model = models.Model(inputs=inp, outputs=out)
    opt = optimizers.Adam(config.learning_rate)

    model.compile(optimizer=opt, loss=losses.categorical_crossentropy, metri
cs=['acc'])
    return model
```

Training 1D Conv

It is important to convert raw labels to integer indices

In [28]:
```
train.head()
```

Out[28]:

	fname	labels	nframes
0	0006ae4e.wav	Bark	310456
1	0019ef41.wav	Raindrop	90616
2	001ec0ad.wav	Finger_snapping	66976
3	0026c7cb.wav	Run	1125886
4	0026f116.wav	Finger_snapping	60638

In [29]:
```
LABELS = list(train.labels.unique())
label_idx = {label: i for i, label in enumerate(LABELS)}
train.set_index("fname", inplace=True)
test.set_index("fname", inplace=True)
train["label_idx"] = train.labels.apply(lambda x: label_idx[x])
if not COMPLETE_RUN:
    train = train[:2000]
    test = test[:2000]
```

In [30]:
```
train.head()
```

Out[30]:

fname	labels	nframes	label_idx
0006ae4e.wav	Bark	310456	0
0019ef41.wav	Raindrop	90616	1
001ec0ad.wav	Finger_snapping	66976	2
0026c7cb.wav	Run	1125886	3
0026f116.wav	Finger_snapping	60638	2

In [31]:
```
config = Config(sampling_rate=16000, audio_duration=2, n_folds=10, learning_
rate=0.001)
if not COMPLETE_RUN:
    config = Config(sampling_rate=100, audio_duration=1, n_folds=2, max_epoc
hs=1)
```

Here is the code for 10-fold training:

- We use **`from sklearn.model_selection.StratifiedKFold`** for splitting the trainig data into 10 folds.
- We use some Keras callbacks to monitor the training.
  - **`ModelCheckpoint`** saves the best weight of our model (using validation data). We use this weight to make test predictions.
  - **`EarlyStopping`** stops the training once validation loss ceases to decrease
  - **`TensorBoard`** helps us visualize training and validation loss and accuracy.
- We fit the model using **`DataGenerator`** for training and validation splits.
- We get both training and test predictions and save them as .npy format. We also generate a submission file. For 10-fold CV, the number of prediction files should be 10. We will ensemble these predictions later.

In [32]:
```
PREDICTION_FOLDER = "predictions_1d_conv"
if not os.path.exists(PREDICTION_FOLDER):
    os.mkdir(PREDICTION_FOLDER)
if os.path.exists('logs/' + PREDICTION_FOLDER):
    shutil.rmtree('logs/' + PREDICTION_FOLDER)

skf = StratifiedKFold(n_splits=config.n_folds)

for i, (train_split, val_split) in enumerate(skf.split(train.index, train.la
bel_idx)):
    train_set = train.iloc[train_split]
    val_set = train.iloc[val_split]
    checkpoint = ModelCheckpoint('best_%d.h5'%i, monitor='val_loss', verbos
e=1, save_best_only=True)
    early = EarlyStopping(monitor="val_loss", mode="min", patience=5)
    tb = TensorBoard(log_dir='./logs/' + PREDICTION_FOLDER + '/fold_%d'%i, w
rite_graph=True)

    callbacks_list = [checkpoint, early, tb]
    print("\nFold: ", i)
    if COMPLETE_RUN:
        model = get_1d_conv_model(config)
    else:
        model = get_1d_dummy_model(config)

    train_generator = DataGenerator(config, '../input/train_curated/', train
_set.index,
                                    train_set.label_idx, batch_size=64,
                                    preprocessing_fn=audio_norm)
    val_generator = DataGenerator(config, '../input/train_curated/', val_se
t.index,
                                  val_set.label_idx, batch_size=64,
                                  preprocessing_fn=audio_norm)

    history = model.fit_generator(train_generator, callbacks=callbacks_list,
validation_data=val_generator,
                                  epochs=config.max_epochs, use_multiprocess
ing=True, max_queue_size=20)

#     model.load_weights('../working/best_%d.h5'%i)

    # Save train predictions
    train_generator = DataGenerator(config, '../input/train_curated/', trai
n.index, batch_size=128,
                                    preprocessing_fn=audio_norm)
    predictions = model.predict_generator(train_generator, use_multiprocessi
ng=True,
                                          max_queue_size=20, verbose=1)
    np.save(PREDICTION_FOLDER + "/train_predictions_%d.npy"%i, predictions)

    # Save test predictions
    test_generator = DataGenerator(config, '../input/test/', test.index, bat
ch_size=128,
                                    preprocessing_fn=audio_norm)
    predictions = model.predict_generator(test_generator, use_multiprocessin
g=True,
                                          max_queue_size=20, verbose=1)
    np.save(PREDICTION_FOLDER + "/test_predictions_%d.npy"%i, predictions)

    # Make a submission file
    top_3 = np.array(LABELS)[np.argsort(-predictions, axis=1)[:, :3]]
    predicted_labels = [' '.join(list(x)) for x in top_3]
    test['label'] = predicted_labels
    test[['label']].to_csv(PREDICTION_FOLDER + "/predictions_%d.csv"%i)
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/model_selection/_split.py:652:
Warning: The least populated class in y has only 1 members, which is too few.
The minimum number of members in any class cannot be less than n_splits=2.
  % (min_groups, self.n_splits)), Warning)

Fold:  0
WARNING:tensorflow:From /opt/conda/lib/python3.6/site-packages/tensorflow/pyt
hon/framework/op_def_library.py:263: colocate_with (from tensorflow.python.fr
amework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From /opt/conda/lib/python3.6/site-packages/tensorflow/pyt
hon/ops/math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is d
eprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Epoch 1/1
16/16 [==============================] - 120s 8s/step - loss: 4.3050 - acc:
0.0127 - val_loss: 4.3058 - val_acc: 0.0128

Epoch 00001: val_loss improved from inf to 4.30585, saving model to best_0.h5
16/16 [==============================] - 233s 15s/step
9/9 [==============================] - 100s 11s/step

Fold:  1
Epoch 1/1
16/16 [==============================] - 109s 7s/step - loss: 4.3089 - acc:
0.0127 - val_loss: 4.3086 - val_acc: 0.0133

Epoch 00001: val_loss improved from inf to 4.30861, saving model to best_1.h5
Epoch 1/1
16/16 [==============================] - 225s 14s/step
9/9 [==============================] - 92s 10s/step
```

predictions are saved as following.

In [33]:
```
os.listdir('../working/predictions_1d_conv/')
```

Out[33]:
```
['predictions_0.csv',
 'train_predictions_1.npy',
 'predictions_1.csv',
 'train_predictions_0.npy',
 'test_predictions_0.npy',
 'test_predictions_1.npy']
```

Ensembling 1D Conv Predictions

Now that we have trained our model, it is time average the predictions of X-folds.
We will try **Geometric Mean averaging**.

In [34]:
```
pred_list = []
for i in range(config.n_folds):
    pred_list.append(np.load("../working/predictions_1d_conv/test_prediction
s_%d.npy"%i))
prediction = np.ones_like(pred_list[0])
for pred in pred_list:
    prediction = prediction*pred
prediction = prediction**(1./len(pred_list))
# Make a submission file
top_3 = np.array(LABELS)[np.argsort(-prediction, axis=1)[:, :3]]
predicted_labels = [' '.join(list(x)) for x in top_3]
test = pd.read_csv('../input/sample_submission.csv')
test['label'] = predicted_labels
test[['fname', 'label']].to_csv("1d_conv_ensembled_submission.csv", index=Fa
lse)
```

3. Introuction to MFCC

As we have seen in the previous section, our Deep Learning models are powerful enough to classify sounds from the raw audio. We do not require any complex feature engineering. But before the Deep Learning era, people developed techniques to extract features from audio signals. It turns out that these techniques are still useful. One such technique is computing the MFCC (Mel Frquency Cepstral Coefficients) from the raw audio. Before we jump to MFCC, let's talk about extracting features from the sound.

If we just want to classify some sound, we should build features that are **speaker independent**. Any feature that only gives information about the speaker (like the pitch of their voice) will not be helpful for classification. In other words, we should extract features that depend on the "content" of the audio rather than the nature of the speaker. Also, a good feature extraction technique should mimic the human speech perception. We don't hear loudness on a linear scale. If we want to double the perceived loudness of a sound, we have to put 8 times as much energy into it. Instead of a linear scale, our perception system uses a log scale.

Taking these things into account, Davis and Mermelstein came up with MFCC in the 1980's. MFCC mimics the logarithmic perception of loudness and pitch of human auditory system and tries to eliminate speaker dependent characteristics by excluding the fundamental frequency and their harmonics. The underlying mathematics is quite complicated and we will skip that. For those interested, here is the [detailed explanation (http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/)](http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/).

Generating MFCC using Librosa

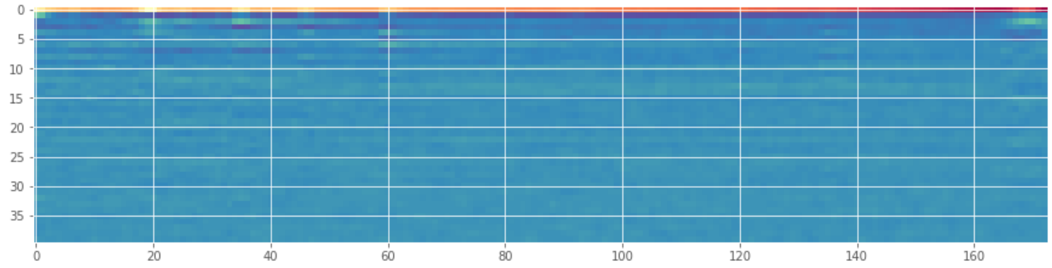
The library librosa has a function to calculate MFCC. Let's compute the MFCC of an audio file and visualize it.

In [35]:
```
import librosa
SAMPLE_RATE = 44100
fname = '../input/train_curated/0019ef41.wav'  # Raindrop
wav, _ = librosa.core.load(fname, sr=SAMPLE_RATE)
wav = wav[:2*44100]
```

In [36]:
```
mfcc = librosa.feature.mfcc(wav, sr = SAMPLE_RATE, n_mfcc=40)
mfcc.shape
```

Out[36]: (40, 173)

In [37]:
```
fig, ax = plt.subplots(figsize=(15, 5))
ax.imshow(mfcc, cmap='Spectral', interpolation='nearest');
```



4. Building a Model using MFCC

We will build a 2D Convolutional model using MFCC.

In [38]:
```
from keras.layers import (Convolution2D, GlobalAveragePooling2D, BatchNormal
ization, Flatten,
                          GlobalMaxPool2D, MaxPool2D, concatenate, Activatio
n)
from keras.utils import Sequence, to_categorical
from keras import backend as K
```

In [39]:
```
def get_2d_dummy_model(config):

    nclass = config.n_classes

    inp = Input(shape=(config.dim[0],config.dim[1],1))
    x = GlobalMaxPool2D()(inp)
    out = Dense(nclass, activation=softmax)(x)

    model = models.Model(inputs=inp, outputs=out)
    opt = optimizers.Adam(config.learning_rate)

    model.compile(optimizer=opt, loss=losses.categorical_crossentropy, metri
cs=['acc'])
    return model


def get_2d_conv_model(config):

    nclass = config.n_classes

    inp = Input(shape=(config.dim[0],config.dim[1],1))
    x = Convolution2D(32, (4,10), padding="same")(inp)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)
    x = MaxPool2D()(x)

    x = Convolution2D(32, (4,10), padding="same")(x)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)
    x = MaxPool2D()(x)

    x = Convolution2D(32, (4,10), padding="same")(x)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)
    x = MaxPool2D()(x)

    x = Convolution2D(32, (4,10), padding="same")(x)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)
    x = MaxPool2D()(x)

    x = Flatten()(x)
    x = Dense(64)(x)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)
    out = Dense(nclass, activation=softmax)(x)

    model = models.Model(inputs=inp, outputs=out)
    opt = optimizers.Adam(config.learning_rate)

    model.compile(optimizer=opt, loss=losses.categorical_crossentropy, metri
cs=['acc'])
    return model
```

Preparing data

In [40]:
```
config = Config(sampling_rate=44100, audio_duration=2, n_folds=10,
                learning_rate=0.001, use_mfcc=True, n_mfcc=40)
if not COMPLETE_RUN:
    config = Config(sampling_rate=44100, audio_duration=2, n_folds=2,
                    max_epochs=1, use_mfcc=True, n_mfcc=40)
```

In [41]:
```
def prepare_data(df, config, data_dir):
    X = np.empty(shape=(df.shape[0], config.dim[0], config.dim[1], 1))
    input_length = config.audio_length
    for i, fname in enumerate(df.index):
#         print(fname)
        file_path = data_dir + fname
        data, _ = librosa.core.load(file_path, sr=config.sampling_rate, res_
type="kaiser_fast")

        # Random offset / Padding
        if len(data) > input_length:
            max_offset = len(data) - input_length
            offset = np.random.randint(max_offset)
            data = data[offset:(input_length+offset)]
        else:
            if input_length > len(data):
                max_offset = input_length - len(data)
                offset = np.random.randint(max_offset)
            else:
                offset = 0
            data = np.pad(data, (offset, input_length - len(data) - offset),
"constant")

        data = librosa.feature.mfcc(data, sr=config.sampling_rate, n_mfcc=co
nfig.n_mfcc)
        data = np.expand_dims(data, axis=-1)
        X[i,] = data
    return X
```

In [42]:
```
test.index = test.fname
```

In [43]:
```
%time
X_train = prepare_data(train, config, '../input/train_curated/')
X_test = prepare_data(test, config, '../input/test/')
y_train = to_categorical(train.label_idx.astype('str'), num_classes=config.n
_classes)
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 23.1 µs
```

Normalization

In [44]:
```
mean = np.mean(X_train, axis=0)
std = np.std(X_train, axis=0)

X_train = (X_train - mean)/std
X_test = (X_test - mean)/std
```

Training 2D Conv on MFCC

In [45]:
```
PREDICTION_FOLDER = "predictions_2d_conv"
if not os.path.exists(PREDICTION_FOLDER):
    os.mkdir(PREDICTION_FOLDER)
if os.path.exists('logs/' + PREDICTION_FOLDER):
    shutil.rmtree('logs/' + PREDICTION_FOLDER)

skf = StratifiedKFold(n_splits=config.n_folds)

for i, (train_split, val_split) in enumerate(skf.split(train.index, train.la
bel_idx)):
    K.clear_session()
    X, y, X_val, y_val = X_train[train_split], y_train[train_split], X_train
[val_split], y_train[val_split]
    checkpoint = ModelCheckpoint('best_%d.h5'%i, monitor='val_loss', verbos
e=1, save_best_only=True)
    early = EarlyStopping(monitor="val_loss", mode="min", patience=5)
    tb = TensorBoard(log_dir='./logs/' + PREDICTION_FOLDER + '/fold_%i'%i, w
rite_graph=True)
    callbacks_list = [checkpoint, early, tb]
    print("#"*50)
    print("Fold: ", i)
    model = get_2d_conv_model(config)
    history = model.fit(X, y, validation_data=(X_val, y_val), callbacks=call
backs_list,
                        batch_size=64, epochs=config.max_epochs)
    model.load_weights('best_%d.h5'%i)

    # Save train predictions
    predictions = model.predict(X_train, batch_size=64, verbose=1)
    np.save(PREDICTION_FOLDER + "/train_predictions_%d.npy"%i, predictions)

    # Save test predictions
    predictions = model.predict(X_test, batch_size=64, verbose=1)
    np.save(PREDICTION_FOLDER + "/test_predictions_%d.npy"%i, predictions)

    # Make a submission file
    top_3 = np.array(LABELS)[np.argsort(-predictions, axis=1)[:, :3]]
    predicted_labels = [' '.join(list(x)) for x in top_3]
    test['label'] = predicted_labels
    test[['label']].to_csv(PREDICTION_FOLDER + "/predictions_%d.csv"%i)
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/model_selection/_split.py:652:
Warning: The least populated class in y has only 1 members, which is too few.
The minimum number of members in any class cannot be less than n_splits=2.
  % (min_groups, self.n_splits)), Warning)

##################################################
Fold:  0
Train on 981 samples, validate on 1019 samples
Epoch 1/1
981/981 [==============================] - 3s 4ms/step - loss: 4.4982 - acc:
0.0183 - val_loss: 4.5130 - val_acc: 0.0343

Epoch 00001: val_loss improved from inf to 4.51296, saving model to best_0.h5
2000/2000 [==============================] - 0s 160us/step
1120/1120 [==============================] - 0s 111us/step
##################################################
Fold:  1
Train on 1019 samples, validate on 981 samples
Epoch 1/1
1019/1019 [==============================] - 1s 1ms/step - loss: 4.4766 - ac
c: 0.0186 - val_loss: 4.4547 - val_acc: 0.0204

Epoch 00001: val_loss improved from inf to 4.45468, saving model to best_1.h5
2000/2000 [==============================] - 0s 155us/step
1120/1120 [==============================] - 0s 103us/step
```

Ensembling 2D Conv Predictions

In [46]:
```
pred_list = []
for i in range(config.n_folds):
    pred_list.append(np.load("../working/predictions_2d_conv/test_prediction
s_%d.npy"%i))
prediction = np.ones_like(pred_list[0])
for pred in pred_list:
    prediction = prediction*pred
prediction = prediction**(1./len(pred_list))
# Make a submission file
top_3 = np.array(LABELS)[np.argsort(-prediction, axis=1)[:, :3]]
predicted_labels = [' '.join(list(x)) for x in top_3]
test = pd.read_csv('../input/sample_submission.csv')
test['label'] = predicted_labels
test[['fname', 'label']].to_csv("2d_conv_ensembled_submission.csv", index=Fa
lse)
```

5. Ensembling 1D Conv and 2D Conv Predictions

In [47]:
```
pred_list = []
for i in range(config.n_folds):
    pred_list.append(np.load("../working/predictions_1d_conv/test_prediction
s_%d.npy"%i))
for i in range(config.n_folds):
    pred_list.append(np.load("../working/predictions_2d_conv/test_prediction
s_%d.npy"%i))
prediction = np.ones_like(pred_list[0])
for pred in pred_list:
    prediction = prediction*pred
prediction = prediction**(1./len(pred_list))
```

Be careful :
Because we exclude multi-labeled records, prediction shape became invalid.

In [48]:
```
prediction.shape
```

Out[48]:
```
(1120, 74)
```

imports

In [1]:
```
import gc
import os
import pickle
import random
import time
from collections import Counter, defaultdict
from functools import partial
from pathlib import Path
from psutil import cpu_count

import librosa
import numpy as np
import pandas as pd
from PIL import Image
from sklearn.model_selection import train_test_split
#from skmultilearn.model_selection import iterative_train_test_split

import torch
import torch.nn as nn
import torch.nn.functional as F
from fastprogress import master_bar, progress_bar
from torch.optim import Adam
from torch.optim.lr_scheduler import CosineAnnealingLR
from torch.utils.data import Dataset, DataLoader
from torchvision.transforms import transforms
```

In [2]:
```
torch.cuda.is_available()
```

Out[2]:
```
True
```

utils

In [3]:
```
def seed_everything(seed):
    random.seed(seed)
    os.environ['PYTHONHASHSEED'] = str(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    torch.backends.cudnn.deterministic = True

SEED = 2019
seed_everything(SEED)
```

In [4]:
```
N_JOBS = cpu_count()
os.environ['MKL_NUM_THREADS'] = str(N_JOBS)
os.environ['OMP_NUM_THREADS'] = str(N_JOBS)
DataLoader = partial(DataLoader, num_workers=N_JOBS)
```

In [5]:
```
# from official code https://colab.research.google.com/drive/1AgPdhSp7ttY180
3fEoHOQKlt_3HJDLi8#scrollTo=cRCaCIb9oguU
def _one_sample_positive_class_precisions(scores, truth):
    """Calculate precisions for each true class for a single sample.

    Args:
      scores: np.array of (num_classes,) giving the individual classifier sc
ores.
      truth: np.array of (num_classes,) bools indicating which classes are t
rue.

    Returns:
      pos_class_indices: np.array of indices of the true classes for this sa
mple.
      pos_class_precisions: np.array of precisions corresponding to each of
those
        classes.
    """
    num_classes = scores.shape[0]
    pos_class_indices = np.flatnonzero(truth > 0)
    # Only calculate precisions if there are some true classes.
    if not len(pos_class_indices):
        return pos_class_indices, np.zeros(0)
    # Retrieval list of classes for this sample.
    retrieved_classes = np.argsort(scores)[::-1]
    # class_rankings[top_scoring_class_index] == 0 etc.
    class_rankings = np.zeros(num_classes, dtype=np.int)
    class_rankings[retrieved_classes] = range(num_classes)
    # Which of these is a true label?
    retrieved_class_true = np.zeros(num_classes, dtype=np.bool)
    retrieved_class_true[class_rankings[pos_class_indices]] = True
    # Num hits for every truncated retrieval list.
    retrieved_cumulative_hits = np.cumsum(retrieved_class_true)
    # Precision of retrieval list truncated at each hit, in order of pos_lab
els.
    precision_at_hits = (
            retrieved_cumulative_hits[class_rankings[pos_class_indices]] /
            (1 + class_rankings[pos_class_indices].astype(np.float)))
    return pos_class_indices, precision_at_hits


def calculate_per_class_lwlrap(truth, scores):
    """Calculate label-weighted label-ranking average precision.

    Arguments:
      truth: np.array of (num_samples, num_classes) giving boolean ground-tr
uth
        of presence of that class in that sample.
      scores: np.array of (num_samples, num_classes) giving the classifier-u
nder-
        test's real-valued score for each class for each sample.

    Returns:
      per_class_lwlrap: np.array of (num_classes,) giving the lwlrap for eac
h
        class.
      weight_per_class: np.array of (num_classes,) giving the prior of each
        class within the truth labels.  Then the overall unbalanced lwlrap i
s
        simply np.sum(per_class_lwlrap * weight_per_class)
    """
    assert truth.shape == scores.shape
    num_samples, num_classes = scores.shape
    # Space to store a distinct precision value for each class on each sampl
e.
    # Only the classes that are true for each sample will be filled in.
    precisions_for_samples_by_classes = np.zeros((num_samples, num_classes))
    for sample_num in range(num_samples):
```

dataset

In [6]:
```
dataset_dir = Path('../input/freesound-audio-tagging-2019')
preprocessed_dir = Path('../input/fat2019_prep_mels1')
```

In [7]:
```
csvs = {
    'train_curated': dataset_dir / 'train_curated.csv',
    #'train_noisy': dataset_dir / 'train_noisy.csv',
    'train_noisy': preprocessed_dir / 'trn_noisy_best50s.csv',
    'sample_submission': dataset_dir / 'sample_submission.csv',
}

dataset = {
    'train_curated': dataset_dir / 'train_curated',
    'train_noisy': dataset_dir / 'train_noisy',
    'test': dataset_dir / 'test',
}

mels = {
    'train_curated': preprocessed_dir / 'mels_train_curated.pkl',
    'train_noisy': preprocessed_dir / 'mels_trn_noisy_best50s.pkl',
    'test': preprocessed_dir / 'mels_test.pkl',  # NOTE: this data doesn't w
ork at 2nd stage
}
```

In [8]:
```
train_curated = pd.read_csv(csvs['train_curated'])
train_noisy = pd.read_csv(csvs['train_noisy'])
train_df = pd.concat([train_curated, train_noisy], sort=True, ignore_index=T
rue)
train_df.head()
```

Out[8]:

	fname	labels	singled
0	0006ae4e.wav	Bark	NaN
1	0019ef41.wav	Raindrop	NaN
2	001ec0ad.wav	Finger_snapping	NaN
3	0026c7cb.wav	Run	NaN
4	0026f116.wav	Finger_snapping	NaN

In [9]:
```
test_df = pd.read_csv(csvs['sample_submission'])
test_df.head()
```

Out[9]:

	fname	Accelerating_and_revving_and_vroom	Accordion	Acoustic_guitar	Applause	Bark	Bass_
0	000ccb97.wav	0	0	0	0	0	
1	0012633b.wav	0	0	0	0	0	
2	001ed5f1.wav	0	0	0	0	0	
3	00294be0.wav	0	0	0	0	0	
4	003fde7a.wav	0	0	0	0	0	

In [10]:
```
labels = test_df.columns[1:].tolist()
labels
```

Out[10]:
```
['Accelerating_and_revving_and_vroom',
 'Accordion',
 'Acoustic_guitar',
 'Applause',
 'Bark',
 'Bass_drum',
 'Bass_guitar',
 'Bathtub_(filling_or_washing)',
 'Bicycle_bell',
 'Burping_and_eructation',
 'Bus',
 'Buzz',
 'Car_passing_by',
 'Cheering',
 'Chewing_and_mastication',
 'Child_speech_and_kid_speaking',
 'Chink_and_clink',
 'Chirp_and_tweet',
 'Church_bell',
 'Clapping',
 'Computer_keyboard',
 'Crackle',
 'Cricket',
 'Crowd',
 'Cupboard_open_or_close',
 'Cutlery_and_silverware',
 'Dishes_and_pots_and_pans',
 'Drawer_open_or_close',
 'Drip',
 'Electric_guitar',
 'Fart',
 'Female_singing',
 'Female_speech_and_woman_speaking',
 'Fill_(with_liquid)',
 'Finger_snapping',
 'Frying_(food)',
 'Gasp',
 'Glockenspiel',
 'Gong',
 'Gurgling',
 'Harmonica',
 'Hi-hat',
 'Hiss',
 'Keys_jangling',
 'Knock',
 'Male_singing',
 'Male_speech_and_man_speaking',
 'Marimba_and_xylophone',
 'Mechanical_fan',
 'Meow',
 'Microwave_oven',
 'Motorcycle',
 'Printer',
 'Purr',
 'Race_car_and_auto_racing',
 'Raindrop',
 'Run',
 'Scissors',
 'Screaming',
 'Shatter',
 'Sigh',
 'Sink_(filling_or_washing)',
 'Skateboard',
 'Slam',
 'Sneeze',
 'Squeak',
 'Stream',
 'Strum',
```

In [11]:
```
num_classes = len(labels)
num_classes
```

Out[11]: 80

In [12]:
```
y_train = np.zeros((len(train_df), num_classes)).astype(int)
for i, row in enumerate(train_df['labels'].str.split(',')):
    for label in row:
        idx = labels.index(label)
        y_train[i, idx] = 1

y_train.shape
```

Out[12]: (8970, 80)

In [13]:
```
with open(mels['train_curated'], 'rb') as curated, open(mels['train_noisy'],
'rb') as noisy:
    x_train = pickle.load(curated)
    x_train.extend(pickle.load(noisy))

with open(mels['test'], 'rb') as test:
    x_test = pickle.load(test)

len(x_train), len(x_test)
```

Out[13]: (8970, 1120)

In [14]:
```
class FATTrainDataset(Dataset):
    def __init__(self, mels, labels, transforms):
        super().__init__()
        self.mels = mels
        self.labels = labels
        self.transforms = transforms

    def __len__(self):
        return len(self.mels)

    def __getitem__(self, idx):
        # crop 1sec
        image = Image.fromarray(self.mels[idx], mode='RGB')
        time_dim, base_dim = image.size
        crop = random.randint(0, time_dim - base_dim)
        image = image.crop([crop, 0, crop + base_dim, base_dim])
        image = self.transforms(image).div_(255)

        label = self.labels[idx]
        label = torch.from_numpy(label).float()

        return image, label
```

In [15]:
```
class FATTestDataset(Dataset):
    def __init__(self, fnames, mels, transforms, tta=5):
        super().__init__()
        self.fnames = fnames
        self.mels = mels
        self.transforms = transforms
        self.tta = tta

    def __len__(self):
        return len(self.fnames) * self.tta

    def __getitem__(self, idx):
        new_idx = idx % len(self.fnames)

        image = Image.fromarray(self.mels[new_idx], mode='RGB')
        time_dim, base_dim = image.size
        crop = random.randint(0, time_dim - base_dim)
        image = image.crop([crop, 0, crop + base_dim, base_dim])
        image = self.transforms(image).div_(255)

        fname = self.fnames[new_idx]

        return image, fname
```

In [16]:
```
transforms_dict = {
    'train': transforms.Compose([
        transforms.RandomHorizontalFlip(0.5),
        transforms.ToTensor(),
    ]),
    'test': transforms.Compose([
        transforms.RandomHorizontalFlip(0.5),
        transforms.ToTensor(),
    ]),
}
```

model

In [17]:
```
class ConvBlock(nn.Module):
    def __init__(self, in_channels, out_channels):
        super().__init__()

        self.conv1 = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, 3, 1, 1),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(),
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(out_channels, out_channels, 3, 1, 1),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(),
        )

        self._init_weights()

    def _init_weights(self):
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                nn.init.kaiming_normal_(m.weight)
                if m.bias is not None:
                    nn.init.zeros_(m.bias)
            elif isinstance(m, nn.BatchNorm2d):
                nn.init.constant_(m.weight, 1)
                nn.init.zeros_(m.bias)

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = F.avg_pool2d(x, 2)
        return x
```

In [18]:
```
class Classifier(nn.Module):
    def __init__(self, num_classes):
        super().__init__()

        self.conv = nn.Sequential(
            ConvBlock(in_channels=3, out_channels=64),
            ConvBlock(in_channels=64, out_channels=128),
            ConvBlock(in_channels=128, out_channels=256),
            ConvBlock(in_channels=256, out_channels=512),
        )

        self.fc = nn.Sequential(
            nn.Dropout(0.2),
            nn.Linear(512, 128),
            nn.PReLU(),
            nn.BatchNorm1d(128),
            nn.Dropout(0.1),
            nn.Linear(128, num_classes),
        )

    def forward(self, x):
        x = self.conv(x)
        x = torch.mean(x, dim=3)
        x, _ = torch.max(x, dim=2)
        x = self.fc(x)
        return x
```

In [19]:
```
Classifier(num_classes=num_classes)
```

Out[19]:
```
Classifier(
  (conv): Sequential(
    (0): ConvBlock(
      (conv1): Sequential(
        (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
        (2): ReLU()
      )
      (conv2): Sequential(
        (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
        (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
        (2): ReLU()
      )
    )
    (1): ConvBlock(
      (conv1): Sequential(
        (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
        (2): ReLU()
      )
      (conv2): Sequential(
        (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
        (2): ReLU()
      )
    )
    (2): ConvBlock(
      (conv1): Sequential(
        (0): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
        (2): ReLU()
      )
      (conv2): Sequential(
        (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
        (2): ReLU()
      )
    )
    (3): ConvBlock(
      (conv1): Sequential(
        (0): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
        (2): ReLU()
      )
      (conv2): Sequential(
        (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
        (2): ReLU()
      )
    )
  )
  (fc): Sequential(
    (0): Dropout(p=0.2)
```

train

In [20]:
```
def train_model(x_train, y_train, train_transforms):
    num_epochs = 80
    batch_size = 64
    test_batch_size = 256
    lr = 3e-3
    eta_min = 1e-5
    t_max = 10

    num_classes = y_train.shape[1]

    x_trn, x_val, y_trn, y_val = train_test_split(x_train, y_train, test_siz
e=0.2, random_state=SEED)

    train_dataset = FATTrainDataset(x_trn, y_trn, train_transforms)
    valid_dataset = FATTrainDataset(x_val, y_val, train_transforms)

    train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=
True)
    valid_loader = DataLoader(valid_dataset, batch_size=test_batch_size, shu
ffle=False)

    model = Classifier(num_classes=num_classes).cuda()
    criterion = nn.BCEWithLogitsLoss().cuda()
    optimizer = Adam(params=model.parameters(), lr=lr, amsgrad=False)
    scheduler = CosineAnnealingLR(optimizer, T_max=t_max, eta_min=eta_min)

    best_epoch = -1
    best_lwlrap = 0.
    mb = master_bar(range(num_epochs))

    for epoch in mb:
        start_time = time.time()
        model.train()
        avg_loss = 0.

        for x_batch, y_batch in progress_bar(train_loader, parent=mb):
            preds = model(x_batch.cuda())
            loss = criterion(preds, y_batch.cuda())

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            avg_loss += loss.item() / len(train_loader)

        model.eval()
        valid_preds = np.zeros((len(x_val), num_classes))
        avg_val_loss = 0.

        for i, (x_batch, y_batch) in enumerate(valid_loader):
            preds = model(x_batch.cuda()).detach()
            loss = criterion(preds, y_batch.cuda())

            preds = torch.sigmoid(preds)
            valid_preds[i * test_batch_size: (i+1) * test_batch_size] = pred
s.cpu().numpy()

            avg_val_loss += loss.item() / len(valid_loader)

        score, weight = calculate_per_class_lwlrap(y_val, valid_preds)
        lwlrap = (score * weight).sum()

        scheduler.step()

        if (epoch + 1) % 5 == 0:
            elapsed = time.time() - start_time
            mb.write(f'Epoch {epoch+1} - avg_train_loss: {avg_loss:.4f}  avg
_val_loss: {avg_val_loss:.4f}  val_lwlrap: {lwlrap:.6f}  time: {elapsed:.0f}
```

In [21]:
```
result = train_model(x_train, y_train, transforms_dict['train'])
```

31.25% [25/80 12:25<27:20]

Epoch 5 - avg_train_loss: 0.0623 avg_val_loss: 0.0768 val_lwlrap: 0.081251 time: 30s

Epoch 10 - avg_train_loss: 0.0561 avg_val_loss: 0.0842 val_lwlrap: 0.145102 time: 30s

Epoch 15 - avg_train_loss: 0.0554 avg_val_loss: 0.2979 val_lwlrap: 0.099892 time: 30s

Epoch 20 - avg_train_loss: 0.0549 avg_val_loss: 0.1380 val_lwlrap: 0.070345 time: 30s

Epoch 25 - avg_train_loss: 0.0501 avg_val_loss: 1.0152 val_lwlrap: 0.084374 time: 30s

43.36% [49/113 00:11<00:15]

In [22]:
```
result
```

Out[22]:
```
{'best_epoch': 72, 'best_lwlrap': 0.6313720751689541}
```

predict

In [23]:
```
def predict_model(test_fnames, x_test, test_transforms, num_classes, *, tta=
5):
    batch_size = 256

    test_dataset = FATTestDataset(test_fnames, x_test, test_transforms, tta=
tta)
    test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=Fa
lse)

    model = Classifier(num_classes=num_classes)
    model.load_state_dict(torch.load('weight_best.pt'))
    model.cuda()
    model.eval()

    all_outputs, all_fnames = [], []

    pb = progress_bar(test_loader)
    for images, fnames in pb:
        preds = torch.sigmoid(model(images.cuda()).detach())
        all_outputs.append(preds.cpu().numpy())
        all_fnames.extend(fnames)

    test_preds = pd.DataFrame(data=np.concatenate(all_outputs),
                              index=all_fnames,
                              columns=map(str, range(num_classes)))
    test_preds = test_preds.groupby(level=0).mean()

    return test_preds
```

In [24]:
```
test_preds = predict_model(test_df['fname'], x_test, transforms_dict['test
'], num_classes, tta=20)
```

100.00% [88/88 00:27<00:00]

In [25]:
```
test_df[labels] = test_preds.values
test_df.to_csv('submission.csv', index=False)
test_df.head()
```

Out[25]:

	fname	Accelerating_and_revving_and_vroom	Accordion	Acoustic_guitar	Applause	Ba
0	000ccb97.wav	0.000001	2.923243e-09	7.161557e-08	1.016948e-06	0.0000
1	0012633b.wav	0.052353	1.996648e-04	2.096502e-03	2.918867e-03	0.0008
2	001ed5f1.wav	0.000172	1.197935e-04	6.463896e-06	1.389198e-03	0.0011
3	00294be0.wav	0.000002	1.181921e-11	4.253022e-08	2.461792e-08	0.0000
4	003fde7a.wav	0.000060	9.370017e-05	2.245676e-05	4.934249e-05	0.0000

Introduction

see V56 for the best result of LB632 -- Finally I beat the current best public kernel using Keras :) -- This probably be my last update on this kernel -- If you find this kernel helpful, please upvote

Version upto V60 have a silly bug of 'if <-- elif' so that model selection is wrong

This is my effort to do a `Keras` replication with comparable baseline to the great kernel of @mhiro2 https://www.kaggle.com/mhiro2/simple-2d-cnn-classifier-with-pytorch (https://www.kaggle.com/mhiro2/simple-2d-cnn-classifier-with-pytorch) (and further improved by @peining), which in turns use the excellent pre-processed data of @daisukelab https://www.kaggle.com/daisukelab/creating-fat2019-preprocessed-data (https://www.kaggle.com/daisukelab/creating-fat2019-preprocessed-data)) -- Note that to inference to the private data in stage-2, you have to preprocess data yourself.

One change I made in a Keras version, in addition to a simple conv net, we can also use a pre-defined architectures [trained from scratch] `MobileNetV2` , `InceptionV3` and `Xception` where you can choose in the kernel. Also, many ideas borrow from a nice kernel of @voglinio https://www.kaggle.com/voglinio/keras-2d-model-5-fold-log-specgram-curated-only (https://www.kaggle.com/voglinio/keras-2d-model-5-fold-log-specgram-curated-only) , I also borrow the SoftMax+BCE loss & TTA ideas from Giba's kernel (BTW, we all know Giba without having to mention his user :).

I apologize that my code is not at all clean; some of the `pytorch` code is still here albeit not used.

Major Updates

- V1 [CV680, LB574]
- V4 [CV66x, LB576]
- V5 [] Add image augmentation module
- V9 [CV679] Add lwlrap TF metric (credit @rio114 : https://www.kaggle.com/rio114/keras-cnn-with-lwlrap-evaluation (https://www.kaggle.com/rio114/keras-cnn-with-lwlrap-evaluation))
- V11 [] Employ list of augmentations mentioned in https://github.com/sainathadapa/kaggle-freesound-audio-tagging/blob/master/approaches_all.md (https://github.com/sainathadapa/kaggle-freesound-audio-tagging/blob/master/approaches_all.md)
- V16 [] Add BCEwithLogits (use only with ACTIVATION = 'linear')
- V17 add SimpleCNN similar to the pytorch baseline
- V22 add Curated-Only, Train-augment options
- V23 add CRNN model
- **V30 LB598 with shallow CNN in 400s, set iteration to 150**
- **V39 LB608 with CoarseDropout Augmentation**
- V40 Simple Snapshot (Checkpoint) Ensemble
- **V52 [CV811, LB616] MixUp+CoarseDropout : credit https://www.kaggle.com/mathormad/resnet50-v2-keras-focal-loss-mix-up (https://www.kaggle.com/mathormad/resnet50-v2-keras-focal-loss-mix-up)**
- **V56 [CV830, LB632] Change Architecture to get the best result**
- V61 fix silly bugs on model selection

In [1]:
```
import gc
import os
import pickle
import random
import time
from collections import Counter, defaultdict
from functools import partial
from pathlib import Path
from psutil import cpu_count
import matplotlib.pyplot as plt

import librosa
import numpy as np
import pandas as pd
from PIL import Image
from sklearn.model_selection import train_test_split
from imgaug import augmenters as iaa
#from skmultilearn.model_selection import iterative_train_test_split

import torch
import torch.nn as nn
import torch.nn.functional as F
from fastprogress import master_bar, progress_bar
from torch.optim import Adam
from torch.optim.lr_scheduler import CosineAnnealingLR
from torch.utils.data import Dataset, DataLoader
from torchvision.transforms import transforms
```

utils

In [2]:
```
NUM_CLASSES = 80
SIZE=128
checkpoint_file = ['model_best1.h5', 'model_best2.h5', 'model_best3.h5']
# See Version40 for 3 snapshots (or you can use only 1 which is normal run)
EPOCHS = [432, 0, 0] #150 for inception, 100 for xception
TTA = [19, 0, 0] #Number of test-time augmentation
BATCH_SIZE = 32

LR = 4e-4
PATIENCE = 10 #ReduceOnPlateau option
LR_FACTOR = 0.8 #ReduceOnPlateau option
CURATED_ONLY = True # use only curated data for training
TRAIN_AUGMENT = True # use augmentation for training data?
VALID_AUGMENT = False
MODEL = 'mobile' #'cnn8th' # choose among 'xception', 'inception', 'mobile',
'crnn', 'simple'
SEED = 520

USE_MIXUP = True
MIXUP_PROB = 0.275

# No K-Fold implementation yet
# NUM_K_FOLDS = 5 # how many folds (K) you gonna splits
# NUM_MODEL_RUN = 5 # how many models (<= K) you gonna train [e.g. set to 1
for a simple train/test split]

# if use BCEwithLogits loss, use Activation = 'linear' only
ACTIVATION = 'linear'
# ACTIVATION = 'softmax'
# ACTIVATION = 'sigmoid'

# LOSS = 'categorical_crossentropy'
# LOSS = 'binary_crossentropy'
LOSS = 'BCEwithLogits'
```

In [3]:
```
def seed_everything(seed):
    random.seed(seed)
    os.environ['PYTHONHASHSEED'] = str(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    torch.backends.cudnn.deterministic = True

seed_everything(SEED)
```

In [4]:
```
# from official code https://colab.research.google.com/drive/1AgPdhSp7ttY180
3fEoHOQKlt_3HJDLi8#scrollTo=cRCaCIb9oguU
def _one_sample_positive_class_precisions(scores, truth):
    """Calculate precisions for each true class for a single sample.

    Args:
      scores: np.array of (num_classes,) giving the individual classifier sc
ores.
      truth: np.array of (num_classes,) bools indicating which classes are t
rue.

    Returns:
      pos_class_indices: np.array of indices of the true classes for this sa
mple.
      pos_class_precisions: np.array of precisions corresponding to each of
those
        classes.
    """
    num_classes = scores.shape[0]
    pos_class_indices = np.flatnonzero(truth > 0)
    # Only calculate precisions if there are some true classes.
    if not len(pos_class_indices):
        return pos_class_indices, np.zeros(0)
    # Retrieval list of classes for this sample.
    retrieved_classes = np.argsort(scores)[::-1]
    # class_rankings[top_scoring_class_index] == 0 etc.
    class_rankings = np.zeros(num_classes, dtype=np.int)
    class_rankings[retrieved_classes] = range(num_classes)
    # Which of these is a true label?
    retrieved_class_true = np.zeros(num_classes, dtype=np.bool)
    retrieved_class_true[class_rankings[pos_class_indices]] = True
    # Num hits for every truncated retrieval list.
    retrieved_cumulative_hits = np.cumsum(retrieved_class_true)
    # Precision of retrieval list truncated at each hit, in order of pos_lab
els.
    precision_at_hits = (
            retrieved_cumulative_hits[class_rankings[pos_class_indices]] /
            (1 + class_rankings[pos_class_indices].astype(np.float)))
    return pos_class_indices, precision_at_hits


def calculate_per_class_lwlrap(truth, scores):
    """Calculate label-weighted label-ranking average precision.

    Arguments:
      truth: np.array of (num_samples, num_classes) giving boolean ground-tr
uth
        of presence of that class in that sample.
      scores: np.array of (num_samples, num_classes) giving the classifier-u
nder-
        test's real-valued score for each class for each sample.

    Returns:
      per_class_lwlrap: np.array of (num_classes,) giving the lwlrap for eac
h
        class.
      weight_per_class: np.array of (num_classes,) giving the prior of each
        class within the truth labels.  Then the overall unbalanced lwlrap i
s
        simply np.sum(per_class_lwlrap * weight_per_class)
    """
    assert truth.shape == scores.shape
    num_samples, num_classes = scores.shape
    # Space to store a distinct precision value for each class on each sampl
e.
    # Only the classes that are true for each sample will be filled in.
    precisions_for_samples_by_classes = np.zeros((num_samples, num_classes))
    for sample_num in range(num_samples):
```

In [5]:
```
import tensorflow as tf



# from https://www.kaggle.com/rio114/keras-cnn-with-lwlrap-evaluation/
def tf_one_sample_positive_class_precisions(y_true, y_pred) :
    num_samples, num_classes = y_pred.shape

    # find true labels
    pos_class_indices = tf.where(y_true > 0)

    # put rank on each element
    retrieved_classes = tf.nn.top_k(y_pred, k=num_classes).indices
    sample_range = tf.zeros(shape=tf.shape(tf.transpose(y_pred)), dtype=tf.i
nt32)
    sample_range = tf.add(sample_range, tf.range(tf.shape(y_pred)[0], delta=
1))
    sample_range = tf.transpose(sample_range)
    sample_range = tf.reshape(sample_range, (-1,num_classes*tf.shape(y_pre
d)[0]))
    retrieved_classes = tf.reshape(retrieved_classes, (-1,num_classes*tf.sha
pe(y_pred)[0]))
    retrieved_class_map = tf.concat((sample_range, retrieved_classes), axis=
0)
    retrieved_class_map = tf.transpose(retrieved_class_map)
    retrieved_class_map = tf.reshape(retrieved_class_map, (tf.shape(y_pre
d)[0], num_classes, 2))

    class_range = tf.zeros(shape=tf.shape(y_pred), dtype=tf.int32)
    class_range = tf.add(class_range, tf.range(num_classes, delta=1))

    class_rankings = tf.scatter_nd(retrieved_class_map,
                                          class_range,
                                          tf.shape(y_pred))

    #pick_up ranks
    num_correct_until_correct = tf.gather_nd(class_rankings, pos_class_indic
es)

    # add one for division for "presicion_at_hits"
    num_correct_until_correct_one = tf.add(num_correct_until_correct, 1)
    num_correct_until_correct_one = tf.cast(num_correct_until_correct_one, t
f.float32)

    # generate tensor [num_sample, predict_rank],
    # top-N predicted elements have flag, N is the number of positive for ea
ch sample.
    sample_label = pos_class_indices[:, 0]
    sample_label = tf.reshape(sample_label, (-1, 1))
    sample_label = tf.cast(sample_label, tf.int32)

    num_correct_until_correct = tf.reshape(num_correct_until_correct, (-1,
1))
    retrieved_class_true_position = tf.concat((sample_label,
                                               num_correct_until_correct), a
xis=1)
    retrieved_pos = tf.ones(shape=tf.shape(retrieved_class_true_positio
n)[0], dtype=tf.int32)
    retrieved_class_true = tf.scatter_nd(retrieved_class_true_position,
                                         retrieved_pos,
                                         tf.shape(y_pred))
    # cumulate predict_rank
    retrieved_cumulative_hits = tf.cumsum(retrieved_class_true, axis=1)

    # find positive position
    pos_ret_indices = tf.where(retrieved_class_true > 0)

    # find cumulative hits
```

In [6]:
```
from keras import backend as k
def BCEwithLogits(y_true, y_pred):
    return K.mean(K.binary_crossentropy(y_true, y_pred, from_logits=True), a
xis=-1)
```

```
Using TensorFlow backend.
```

dataset

In [7]:
```
dataset_dir = Path('../input/freesound-audio-tagging-2019')
preprocessed_dir = Path('../input/fat2019_prep_mels1')
```

In [8]:
```
csvs = {
    'train_curated': dataset_dir / 'train_curated.csv',
    #'train_noisy': dataset_dir / 'train_noisy.csv',
    'train_noisy': preprocessed_dir / 'trn_noisy_best50s.csv',
    'sample_submission': dataset_dir / 'sample_submission.csv',
}

dataset = {
    'train_curated': dataset_dir / 'train_curated',
    'train_noisy': dataset_dir / 'train_noisy',
    'test': dataset_dir / 'test',
}

mels = {
    'train_curated': preprocessed_dir / 'mels_train_curated.pkl',
    'train_noisy': preprocessed_dir / 'mels_trn_noisy_best50s.pkl',
    'test': preprocessed_dir / 'mels_test.pkl',  # NOTE: this data doesn't w
ork at 2nd stage
}
```

In [9]:
```
train_curated = pd.read_csv(csvs['train_curated'])
train_noisy = pd.read_csv(csvs['train_noisy'])
if CURATED_ONLY:
    train_df = train_curated
else:
    train_df = pd.concat([train_curated, train_noisy], sort=True, ignore_ind
ex=True)
train_df.head()
```

Out[9]:

	fname	labels
0	0006ae4e.wav	Bark
1	0019ef41.wav	Raindrop
2	001ec0ad.wav	Finger_snapping
3	0026c7cb.wav	Run
4	0026f116.wav	Finger_snapping

In [10]:
```
test_df = pd.read_csv(csvs['sample_submission'])
test_df.head()
```

Out[10]:

	fname	Accelerating_and_revving_and_vroom	Accordion	Acoustic_guitar	Applause	Bark	Bass_
0	000ccb97.wav	0	0	0	0	0	
1	0012633b.wav	0	0	0	0	0	
2	001ed5f1.wav	0	0	0	0	0	
3	00294be0.wav	0	0	0	0	0	
4	003fde7a.wav	0	0	0	0	0	

In [11]:
```
labels = test_df.columns[1:].tolist()
labels[:10]
```

Out[11]:
```
['Accelerating_and_revving_and_vroom',
 'Accordion',
 'Acoustic_guitar',
 'Applause',
 'Bark',
 'Bass_drum',
 'Bass_guitar',
 'Bathtub_(filling_or_washing)',
 'Bicycle_bell',
 'Burping_and_eructation']
```

In [12]:
```
num_classes = len(labels)
num_classes
```

Out[12]:
```
80
```

In [13]:
```
y_train = np.zeros((len(train_df), num_classes)).astype(int)
for i, row in enumerate(train_df['labels'].str.split(',')):
    for label in row:
        idx = labels.index(label)
        y_train[i, idx] = 1

y_train.shape
```

Out[13]:
```
(4970, 80)
```

In [14]:
```
with open(mels['train_curated'], 'rb') as curated, open(mels['train_noisy'],
'rb') as noisy:
    x_train = pickle.load(curated)
    if CURATED_ONLY == False:
        x_train.extend(pickle.load(noisy))

with open(mels['test'], 'rb') as test:
    x_test = pickle.load(test)

len(x_train), len(x_test)
```

Out[14]:
```
(4970, 1120)
```

In [15]:
```
for ii in range(5):
    print(x_train[ii].shape) #x_train is of shape (TRAIN_NUM,128,LEN,3) [4D Tensor]
    print(x_test[ii].shape,'\n')  #x_test of shape (TEST_NUM,128,LEN,3) [4D Tensor]
```

```
(128, 448, 3)
(128, 128, 3)

(128, 131, 3)
(128, 1021, 3)

(128, 128, 3)
(128, 300, 3)

(128, 1623, 3)
(128, 1146, 3)

(128, 128, 3)
(128, 1442, 3)
```

model

In [16]:
```
from keras.layers import *
from keras.models import Sequential, load_model, Model
from keras import metrics
from keras.optimizers import Adam
from keras import backend as K
import keras
from keras.models import Model
from keras.applications.inception_v3 import InceptionV3
from keras.applications.inception_v3 import preprocess_input as preprocess_i
nception
from keras.applications.mobilenet_v2 import MobileNetV2
from keras.applications.mobilenet_v2 import preprocess_input as preprocess_m
obile
from keras.applications.xception import Xception
from keras.applications.xception import preprocess_input as preprocess_xcept
ion

from keras.utils import Sequence
from sklearn.utils import shuffle
def create_model_inception(n_out=NUM_CLASSES):

    base_model =InceptionV3(weights=None, include_top=False)

    x0 = base_model.output
    x1 = GlobalAveragePooling2D()(x0)
    x2 = GlobalMaxPooling2D()(x0)
    x = Concatenate()([x1,x2])

    x = BatchNormalization()(x)
    x = Dropout(0.5)(x)

    x = Dense(256, activation='relu')(x)
    x = BatchNormalization()(x)
    x = Dropout(0.5)(x)


    predictions = Dense(n_out, activation=ACTIVATION)(x)

    # this is the model we will train
    model = Model(inputs=base_model.input, outputs=predictions)
    return model
```

In [17]:
```
def create_model_xception(n_out=NUM_CLASSES):

    base_model = Xception(weights=None, include_top=False)

    x0 = base_model.output
    x1 = GlobalAveragePooling2D()(x0)
    x2 = GlobalMaxPooling2D()(x0)
    x = Concatenate()([x1,x2])

    x = BatchNormalization()(x)
    x = Dropout(0.5)(x)

    x = Dense(256, activation='relu')(x)
    x = BatchNormalization()(x)
    x = Dropout(0.5)(x)

#     x = Dense(128, activation='relu')(x)
#     x = BatchNormalization()(x)
#     x = Dropout(0.3)(x)

    predictions = Dense(n_out, activation=ACTIVATION)(x)

    # this is the model we will train
    model = Model(inputs=base_model.input, outputs=predictions)
    return model
```

In [18]:
```
def create_model_mobile(n_out=NUM_CLASSES):

    base_model =MobileNetV2(weights=None, include_top=False)

    x0 = base_model.output
    x1 = GlobalAveragePooling2D()(x0)
    x2 = GlobalMaxPooling2D()(x0)
    x = Concatenate()([x1,x2])

    x = BatchNormalization()(x)
    x = Dropout(0.5)(x)

    x = Dense(256, activation='relu')(x)
    x = BatchNormalization()(x)
    x = Dropout(0.5)(x)

#     x = Dense(128, activation='relu')(x)
#     x = BatchNormalization()(x)
#     x = Dropout(0.25)(x)


    predictions = Dense(n_out, activation=ACTIVATION)(x)

    # this is the model we will train
    model = Model(inputs=base_model.input, outputs=predictions)
    return model
```

In [19]:
```
def conv_simple_block(x, n_filters):

    x = Convolution2D(n_filters, (3,1), padding="same")(x)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)

    x = Convolution2D(n_filters, (3,1), padding="same")(x)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)
    x = AveragePooling2D()(x)

    return x

def create_model_simplecnn(n_out=NUM_CLASSES):

    inp = Input(shape=(128,128,3))
#     inp = Input(shape=(None,None,3))
    x = conv_simple_block(inp,64)
    x = conv_simple_block(x,128)
    x = conv_simple_block(x,256)
    x = conv_simple_block(x,128)

#     x1 = GlobalAveragePooling2D()(x)
#     x2 = GlobalMaxPooling2D()(x)
#     x = Add()([x1,x2])

    x = Flatten()(x)
    x = Dropout(0.2)(x)

    x = Dense(128, activation='linear')(x)
    x = PReLU()(x)
    x = BatchNormalization()(x)
    x = Dropout(0.2)(x)
    predictions = Dense(n_out, activation=ACTIVATION)(x)

    model = Model(inputs=inp, outputs=predictions)
    return model
```

In [20]:
```
def output_of_lambda(input_shape):
    return (input_shape[0], input_shape[2], input_shape[3])

def my_max(x):
    return K.max(x, axis=1, keepdims=False)

def crnn_simple_block(x, n_filters):

    x = Convolution2D(n_filters, (3,1), padding="same")(x)
    x = Activation("relu")(x)

    x = Convolution2D(n_filters, (3,1), padding="same")(x)
    x = Activation("relu")(x)
    x = MaxPooling2D()(x)
    x = Dropout(0.2)(x)

    return x

def create_model_crnn(n_out=NUM_CLASSES):

#     inp = Input(shape=(128,128,3))
    inp = Input(shape=(128,None,3))
    x = crnn_simple_block(inp,64)
    x = crnn_simple_block(x,128)
    x = crnn_simple_block(x,256)

    # eliminate the frequency dimension, x = (batch, time, channels)
    x = Lambda(my_max, output_shape=output_of_lambda)(x)

    x = Bidirectional(CuDNNGRU(128, return_sequences=True))(x)
#     x = Bidirectional(CuDNNLSTM(64, return_sequences=True))(x)
    x = GlobalMaxPooling1D()(x)
    x = Dense(128, activation='linear')(x)
    x = PReLU()(x)
    x = BatchNormalization()(x)
    x = Dropout(0.2)(x)
    predictions = Dense(n_out, activation=ACTIVATION)(x)

    model = Model(inputs=inp, outputs=predictions)
    return model
```

In [21]:
```
# from the 8th solution in 2018 competition
# https://github.com/sainathadapa/kaggle-freesound-audio-tagging
def create_model_cnn8th(n_out=NUM_CLASSES):
    regu=0
    inp = Input(shape=(128,128,3))

    x = Conv2D(48, 11,  strides=(1,1),kernel_initializer='he_uniform', activ
ation='relu', padding='same',kernel_regularizer=regularizers.l2(regu))(inp)
    x = BatchNormalization()(x)
    x = Conv2D(48, 11,  strides=(2,3),kernel_initializer='he_uniform', activ
ation='relu', padding='same',kernel_regularizer=regularizers.l2(regu))(x)
    x = MaxPooling2D(3, strides=(1,2))(x)
    x = BatchNormalization()(x)

    x = Conv2D(128, 5, strides=(1,1),kernel_initializer='he_uniform', activa
tion='relu', padding='same',kernel_regularizer=regularizers.l2(regu))(x)
    x = BatchNormalization()(x)
    x = Conv2D(128, 5, strides=(2,3),kernel_initializer='he_uniform', activa
tion='relu', padding='same',kernel_regularizer=regularizers.l2(regu))(x)
    x = MaxPooling2D(3, strides=2)(x)
    x = BatchNormalization()(x)

    x = Conv2D(192, 3, strides=1,kernel_initializer='he_uniform', activatio
n='relu', padding='same')(x)
    x = BatchNormalization()(x)
    x = Conv2D(192, 3, strides=1,kernel_initializer='he_uniform', activatio
n='relu', padding='same')(x)
    x = BatchNormalization()(x)
    x = Conv2D(128, 3, strides=1,kernel_initializer='he_uniform', activatio
n='relu', padding='same',kernel_regularizer=regularizers.l2(regu))(x)
    x = MaxPooling2D(3, strides=(1,2))(x)
    x = BatchNormalization()(x)

    x = Flatten()(x)
    x = Dense(256, activation='relu')(x)
    x = Dropout(0.5)(x)
    x = Dense(256, activation='relu')(x)
    x = Dropout(0.5)(x)

    predictions = Dense(n_out, activation=ACTIVATION)(x)

    model = Model(inputs=inp, outputs=predictions)
    return model
```

In [22]:
```
K.clear_session()
'''Choose your model here'''
if MODEL == 'xception':
    preprocess_input = preprocess_xception
    model = create_model_xception(n_out=NUM_CLASSES)
elif MODEL == 'inception':
    preprocess_input = preprocess_inception
    model = create_model_inception(n_out=NUM_CLASSES)
elif MODEL == 'mobile':
    preprocess_input = preprocess_mobile
    model = create_model_mobile(n_out=NUM_CLASSES)
elif MODEL == 'crnn':
    preprocess_input = preprocess_mobile
    model = create_model_crnn(n_out=NUM_CLASSES)
elif MODEL == 'cnn8th':
    preprocess_input = preprocess_mobile
    model = create_model_cnn8th(n_out=NUM_CLASSES)
else:
    preprocess_input = preprocess_mobile
    model = create_model_simplecnn(n_out=NUM_CLASSES)

print(MODEL)
model.summary()
```

```
WARNING:tensorflow:From /opt/conda/lib/python3.6/site-packages/tensorflow/pyt
hon/framework/op_def_library.py:263: colocate_with (from tensorflow.python.fr
amework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From /opt/conda/lib/python3.6/site-packages/keras/backend/
tensorflow_backend.py:3445: calling dropout (from tensorflow.python.ops.nn_op
s) with keep_prob is deprecated and will be removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - k
eep_prob`.
mobile
______________________________________________________________________________
_____________________
Layer (type)                     Output Shape         Param #     Connected to
==============================================================================
=====================
input_1 (InputLayer)             (None, None, None, 3 0
______________________________________________________________________________
_____________________
Conv1_pad (ZeroPadding2D)        (None, None, None, 3 0           input_1
[0][0]
______________________________________________________________________________
_____________________
Conv1 (Conv2D)                   (None, None, None, 3 864         Conv1_pad
[0][0]
______________________________________________________________________________
_____________________
bn_Conv1 (BatchNormalization)    (None, None, None, 3 128         Conv1[0][0]
______________________________________________________________________________
_____________________
Conv1_relu (ReLU)                (None, None, None, 3 0           bn_Conv1
[0][0]
______________________________________________________________________________
_____________________
expanded_conv_depthwise (Depthw (None, None, None, 3 288         Conv1_relu
[0][0]
______________________________________________________________________________
_____________________
expanded_conv_depthwise_BN (Bat (None, None, None, 3 128         expanded_con
v_depthwise[0][0]
______________________________________________________________________________
_____________________
expanded_conv_depthwise_relu (R (None, None, None, 3 0           expanded_con
v_depthwise_BN[0][0]
______________________________________________________________________________
_____________________
expanded_conv_project (Conv2D)  (None, None, None, 1 512         expanded_con
v_depthwise_relu[0][0
______________________________________________________________________________
_____________________
expanded_conv_project_BN (Batch (None, None, None, 1 64          expanded_con
v_project[0][0]
______________________________________________________________________________
_____________________
block_1_expand (Conv2D)          (None, None, None, 9 1536        expanded_con
v_project_BN[0][0]
______________________________________________________________________________
_____________________
block_1_expand_BN (BatchNormali (None, None, None, 9 384         block_1_expa
nd[0][0]
______________________________________________________________________________
_____________________
block_1_expand_relu (ReLU)       (None, None, None, 9 0           block_1_expa
nd_BN[0][0]
______________________________________________________________________________
_____________________
block_1_pad (ZeroPadding2D)      (None, None, None, 9 0           block_1_expa
```

train

In [23]:

```
import numpy as np
xx = np.random.rand(1)
print(xx.shape,xx)

xx = np.random.rand(1,1)
print(xx.shape)
```

```
(1,) [0.70582626]
(1, 1)
```

In [24]:

```
```

In [24]:
```
# If you want, you can try more advanced augmentation like this
augment_img = iaa.Sequential([
#         iaa.ContrastNormalization((0.9, 1.1)),
#         iaa.Multiply((0.9, 1.1), per_channel=0.2),
        iaa.Fliplr(0.5),
#         iaa.GaussianBlur(sigma=(0, 0.1)),
#         iaa.Affine( # x-shift
#             translate_percent={"x": (-0.1, 0.1), "y": (-0.0, 0.0)},
#         ),
        iaa.CoarseDropout(0.12,size_percent=0.05) # see examples : https://g
ithub.com/aleju/imgaug
            ], random_order=True)



# Or you can choose this simplest augmentation (like pytorch version)
# augment_img = iaa.Fliplr(0.5)

# This is my ugly modification; sorry about that
class FATTrainDataset(Sequence):

    def mix_up(x, y):
        x = np.array(x, np.float32)
        lam = np.random.beta(1.0, 1.0)
        ori_index = np.arange(int(len(x)))
        index_array = np.arange(int(len(x)))
        np.random.shuffle(index_array)

        mixed_x = lam * x[ori_index] + (1 - lam) * x[index_array]
        mixed_y = lam * y[ori_index] + (1 - lam) * y[index_array]

        return mixed_x, mixed_y

    def getitem(image):
        # crop 2sec

        base_dim, time_dim, _ = image.shape
        crop = random.randint(0, time_dim - base_dim)
        image = image[:,crop:crop+base_dim,:]

        image = preprocess_input(image)

#         label = self.labels[idx]
        return image
    def create_generator(train_X, train_y, batch_size, shape, augument=Fals
e, shuffling=False, test_data=False, mixup=False, mixup_prob=0.3):
        assert shape[2] == 3
        while True:
            if shuffling:
                train_X,train_y = shuffle(train_X,train_y)

            for start in range(0, len(train_y), batch_size):
                end = min(start + batch_size, len(train_y))
                batch_images = []
                X_train_batch = train_X[start:end]
                if test_data == False:
                    batch_labels = train_y[start:end]

                for i in range(len(X_train_batch)):
                    image = FATTrainDataset.getitem(X_train_batch[i])
                    if augument:
                        image = FATTrainDataset.augment(image)
                    batch_images.append(image)

                if (mixup and test_data == False):
                    dice = np.random.rand(1)
                    if dice > mixup_prob:
                        batch_images, batch_labels =  FATTrainDataset.mix_up
```

In [25]:
```
from keras.callbacks import (ModelCheckpoint, LearningRateScheduler,
                             EarlyStopping, ReduceLROnPlateau,CSVLogger)

from sklearn.model_selection import train_test_split,KFold


reduceLROnPlat = ReduceLROnPlateau(monitor='val_tf_lwlrap', factor=LR_FACTO
R, patience=PATIENCE,
                                   verbose=1, mode='max', min_delta=0.0001,
cooldown=2, min_lr=1e-5 )

csv_logger = CSVLogger(filename='../working/training_log.csv',
                       separator=',',
                       append=True)

checkpoint = ModelCheckpoint(checkpoint_file[0], monitor='val_tf_lwlrap', ve
rbose=1,
                             save_best_only=True, mode='max', save_weights_o
nly = False)
callbacks_list = [checkpoint, csv_logger, reduceLROnPlat]
```

In [26]:
```
# split data into train, valid
x_trn, x_val, y_trn, y_val = train_test_split(x_train, y_train, test_size=0.
2, random_state=SEED)

# create train and valid datagens
train_generator = FATTrainDataset.create_generator(
    x_trn, y_trn, BATCH_SIZE, (SIZE,SIZE,3), augument=TRAIN_AUGMENT, shuffli
ng=True, mixup = USE_MIXUP, mixup_prob = MIXUP_PROB)
validation_generator = FATTrainDataset.create_generator(
    x_val, y_val, BATCH_SIZE, (SIZE,SIZE,3), augument=VALID_AUGMENT, shuffli
ng=False)
```

In [27]:
```
train_steps = np.ceil(float(len(x_trn)) / float(BATCH_SIZE))
val_steps = np.ceil(float(len(x_val)) / float(BATCH_SIZE))
train_steps = train_steps.astype(int)
val_steps = val_steps.astype(int)
print(train_steps, val_steps)
print(len(x_trn))
```

```
125 32
3976
```

In [28]:
```
print(LOSS)
if LOSS=='BCEwithLogits':
     model.compile(loss=BCEwithLogits,
            optimizer=Adam(lr=LR),
            metrics=[tf_lwlrap,'categorical_accuracy'])
else:
    model.compile(loss=LOSS,
            optimizer=Adam(lr=LR),
            metrics=[tf_lwlrap,'categorical_accuracy'])
```

```
BCEwithLogits
```

In [29]:
```
print(LR, PATIENCE, LR_FACTOR,BATCH_SIZE, TRAIN_AUGMENT, USE_MIXUP, MIXUP_PR
OB)
```

```
0.0004 10 0.8 32 True True 0.275
```

In [30]:
```
hist = model.fit_generator(
    train_generator,
    steps_per_epoch=train_steps,
    validation_data=validation_generator,
    validation_steps=val_steps,
    epochs=EPOCHS[0],
    verbose=1,
    callbacks=callbacks_list)
```

```
WARNING:tensorflow:From /opt/conda/lib/python3.6/site-packages/tensorflow/pyt
hon/ops/math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is d
eprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
WARNING:tensorflow:From /opt/conda/lib/python3.6/site-packages/tensorflow/pyt
hon/ops/math_grad.py:102: div (from tensorflow.python.ops.math_ops) is deprec
ated and will be removed in a future version.
Instructions for updating:
Deprecated in favor of operator or tf.math.divide.
Epoch 1/432
125/125 [==============================] - 30s 238ms/step - loss: 0.8264 - tf
_lwlrap: 0.0999 - categorical_accuracy: 0.0133 - val_loss: 0.6638 - val_tf_lw
lrap: 0.0758 - val_categorical_accuracy: 0.0141

Epoch 00001: val_tf_lwlrap improved from -inf to 0.07578, saving model to mod
el_best1.h5
Epoch 2/432
125/125 [==============================] - 16s 129ms/step - loss: 0.6238 - tf
_lwlrap: 0.0978 - categorical_accuracy: 0.0090 - val_loss: 0.4021 - val_tf_lw
lrap: 0.0717 - val_categorical_accuracy: 0.0221

Epoch 00002: val_tf_lwlrap did not improve from 0.07578
Epoch 3/432
125/125 [==============================] - 16s 129ms/step - loss: 0.3651 - tf
_lwlrap: 0.0926 - categorical_accuracy: 0.0103 - val_loss: 0.1388 - val_tf_lw
lrap: 0.0615 - val_categorical_accuracy: 0.0080

Epoch 00003: val_tf_lwlrap did not improve from 0.07578
Epoch 4/432
125/125 [==============================] - 16s 130ms/step - loss: 0.1845 - tf
_lwlrap: 0.0982 - categorical_accuracy: 0.0148 - val_loss: 0.1091 - val_tf_lw
lrap: 0.0715 - val_categorical_accuracy: 0.0231

Epoch 00004: val_tf_lwlrap did not improve from 0.07578
Epoch 5/432
125/125 [==============================] - 16s 129ms/step - loss: 0.1222 - tf
_lwlrap: 0.0999 - categorical_accuracy: 0.0108 - val_loss: 0.1082 - val_tf_lw
lrap: 0.0747 - val_categorical_accuracy: 0.0191

Epoch 00005: val_tf_lwlrap did not improve from 0.07578
Epoch 6/432
125/125 [==============================] - 16s 129ms/step - loss: 0.0965 - tf
_lwlrap: 0.1110 - categorical_accuracy: 0.0168 - val_loss: 0.0800 - val_tf_lw
lrap: 0.0819 - val_categorical_accuracy: 0.0161

Epoch 00006: val_tf_lwlrap improved from 0.07578 to 0.08190, saving model to
model_best1.h5
Epoch 7/432
125/125 [==============================] - 16s 130ms/step - loss: 0.0881 - tf
_lwlrap: 0.1187 - categorical_accuracy: 0.0205 - val_loss: 0.0843 - val_tf_lw
lrap: 0.0866 - val_categorical_accuracy: 0.0191

Epoch 00007: val_tf_lwlrap improved from 0.08190 to 0.08659, saving model to
model_best1.h5
Epoch 8/432
125/125 [==============================] - 17s 134ms/step - loss: 0.0851 - tf
_lwlrap: 0.1251 - categorical_accuracy: 0.0213 - val_loss: 0.1017 - val_tf_lw
lrap: 0.0875 - val_categorical_accuracy: 0.0211

Epoch 00008: val_tf_lwlrap improved from 0.08659 to 0.08750, saving model to
model_best1.h5
Epoch 9/432
125/125 [==============================] - 16s 129ms/step - loss: 0.0835 - tf
_lwlrap: 0.1282 - categorical_accuracy: 0.0238 - val_loss: 0.0800 - val_tf_lw
lrap: 0.0975 - val_categorical_accuracy: 0.0211

Epoch 00009: val_tf_lwlrap improved from 0.08750 to 0.09754, saving model to
```

In [31]:
```
print(K.eval(model.optimizer.lr))
```

```
1e-05
```

In [32]:
```
# if LOSS=='BCEwithLogits':
#      model.compile(loss=BCEwithLogits,
#             optimizer=Adam(lr=3e-4),
#             metrics=[tf_lwlrap,'categorical_accuracy'])
# else:
#     model.compile(loss=LOSS,
#             optimizer=Adam(lr=3e-4),
#             metrics=[tf_lwlrap,'categorical_accuracy'])

# train_generator = FATTrainDataset.create_generator(
#     x_trn, y_trn, BATCH_SIZE, (SIZE,SIZE,3), augument=TRAIN_AUGMENT,
#     shuffling=True, mixup = False, mixup_prob=0.1)

# EPOCHS = [100, 66, 0]

# print(K.eval(model.optimizer.lr))
```

In [33]:
```
if EPOCHS[1] > 0:
    checkpoint = ModelCheckpoint(checkpoint_file[1], monitor='val_tf_lwlrap
', verbose=1,
                             save_best_only=True, mode='max', save_weights_o
nly = False)
    callbacks_list = [checkpoint, csv_logger, reduceLROnPlat]

    hist = model.fit_generator(
    train_generator,
    steps_per_epoch=train_steps,
    validation_data=validation_generator,
    validation_steps=val_steps,
    epochs=EPOCHS[1],
    verbose=1,
    callbacks=callbacks_list)
```

In [34]:
```
print(K.eval(model.optimizer.lr))
```

```
1e-05
```

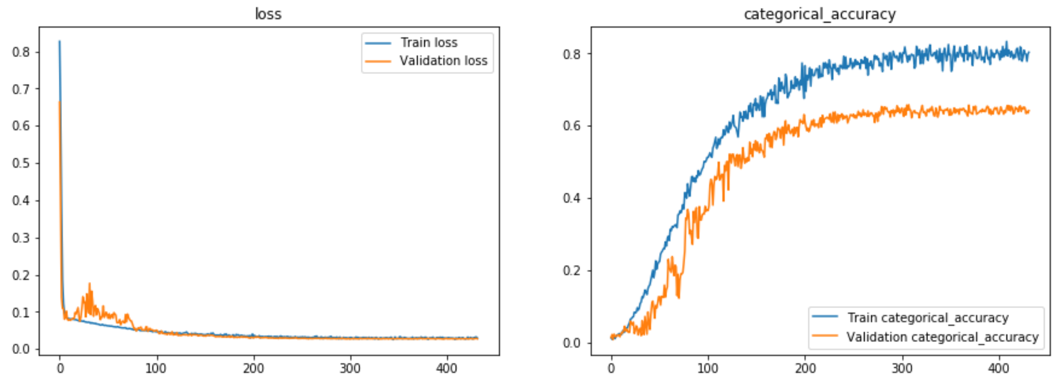
In [35]:
```
if EPOCHS[2] > 0:
    checkpoint = ModelCheckpoint(checkpoint_file[2], monitor='val_tf_lwlrap
', verbose=1,
                             save_best_only=True, mode='max', save_weights_o
nly = False)
    callbacks_list = [checkpoint, csv_logger, reduceLROnPlat]

    hist = model.fit_generator(
    train_generator,
    steps_per_epoch=train_steps,
    validation_data=validation_generator,
    validation_steps=val_steps,
    epochs=EPOCHS[2],
    verbose=1,
    callbacks=callbacks_list)
```

In [36]:
```
print(K.eval(model.optimizer.lr))
```

```
1e-05
```

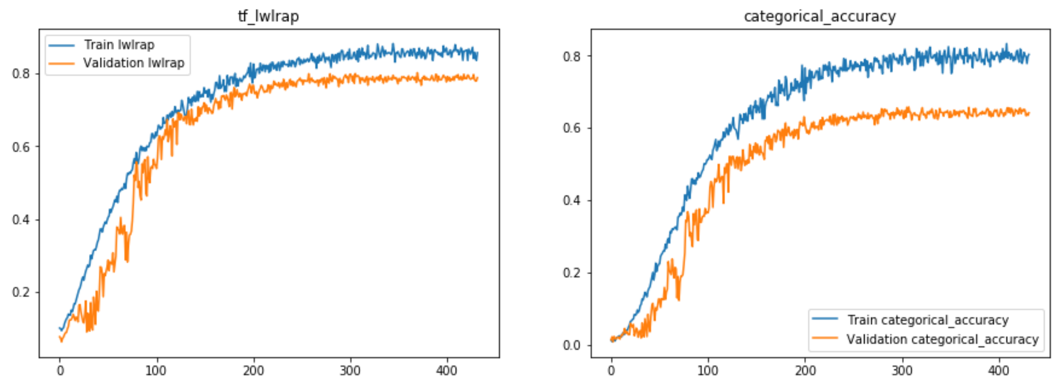
In [37]:
```
fig, ax = plt.subplots(1, 2, figsize=(15,5))
ax[0].set_title('loss')
ax[0].plot(hist.epoch, hist.history["loss"], label="Train loss")
ax[0].plot(hist.epoch, hist.history["val_loss"], label="Validation loss")
ax[1].set_title('categorical_accuracy')
ax[1].plot(hist.epoch, hist.history["categorical_accuracy"], label="Train ca
tegorical_accuracy")
ax[1].plot(hist.epoch, hist.history["val_categorical_accuracy"], label="Vali
dation categorical_accuracy")
ax[0].legend()
ax[1].legend()
```

Out[37]: <matplotlib.legend.Legend at 0x7fd0e9d46898>



In [38]:
```
fig, ax = plt.subplots(1, 2, figsize=(15,5))
ax[0].set_title('tf_lwlrap')
ax[0].plot(hist.epoch, hist.history["tf_lwlrap"], label="Train lwlrap")
ax[0].plot(hist.epoch, hist.history["val_tf_lwlrap"], label="Validation lwlr
ap")
ax[1].set_title('categorical_accuracy')
ax[1].plot(hist.epoch, hist.history["categorical_accuracy"], label="Train ca
tegorical_accuracy")
ax[1].plot(hist.epoch, hist.history["val_categorical_accuracy"], label="Vali
dation categorical_accuracy")
ax[0].legend()
ax[1].legend()
```

Out[38]: <matplotlib.legend.Legend at 0x7fd4b26e4e48>



In [39]:
```
```

Calculate Validation Score using TTA

Note that we have to initiate validation_generation everytime before doing a new prediction as `model.fit_generator` will mis-index examples at the end of epoch (and you will get random score)

In [39]:
```
model.load_weights(checkpoint_file[0])

validation_generator = FATTrainDataset.create_generator(
      x_val, y_val, BATCH_SIZE, (SIZE,SIZE,3), augument=False, shuffling=Fal
se)
pred_val_y = model.predict_generator(validation_generator,steps=val_steps,ve
rbose=1)

for kk in range(len(TTA)):

    for ii in range(TTA[kk]):
        validation_generator = FATTrainDataset.create_generator(
          x_val, y_val, BATCH_SIZE, (SIZE,SIZE,3), augument=False, shufflin
g=False)

        pred_val_y += model.predict_generator(validation_generator,steps=val
_steps,verbose=1)

    if kk+1 < len(TTA) and TTA[kk+1] > 0:
        model.load_weights(checkpoint_file[kk+1])

'''Since the score is based on ranking, we do not need to normalize the pred
iction'''
# pred_val_y = pred_val_y/10
```

```
32/32 [==============================] - 2s 52ms/step
32/32 [==============================] - 0s 15ms/step
32/32 [==============================] - 1s 16ms/step
32/32 [==============================] - 0s 15ms/step
32/32 [==============================] - 1s 16ms/step
32/32 [==============================] - 1s 16ms/step
32/32 [==============================] - 0s 16ms/step
32/32 [==============================] - 1s 16ms/step
32/32 [==============================] - 1s 16ms/step
32/32 [==============================] - 1s 16ms/step
32/32 [==============================] - 1s 16ms/step
32/32 [==============================] - 0s 16ms/step
32/32 [==============================] - 1s 16ms/step
32/32 [==============================] - 1s 16ms/step
32/32 [==============================] - 1s 16ms/step
32/32 [==============================] - 1s 16ms/step
32/32 [==============================] - 1s 16ms/step
32/32 [==============================] - 1s 16ms/step
32/32 [==============================] - 1s 16ms/step
32/32 [==============================] - 1s 16ms/step
```

Out[39]:
```
'Since the score is based on ranking, we do not need to normalize the predict
ion'
```

In [40]:
```
train_generator = FATTrainDataset.create_generator(
    x_trn, y_trn, BATCH_SIZE, (SIZE,SIZE,3), augument=False, shuffling=Fals
e)
pred_train_y = model.predict_generator(train_generator,steps=train_steps,ver
bose=1)
```

```
125/125 [==============================] - 2s 15ms/step
```

In [41]:
```
import sklearn.metrics
def calculate_overall_lwlrap_sklearn(truth, scores):
    """Calculate the overall lwlrap using sklearn.metrics.lrap."""
    # sklearn doesn't correctly apply weighting to samples with no labels, s
o just skip them.
    sample_weight = np.sum(truth > 0, axis=1)
    nonzero_weight_sample_indices = np.flatnonzero(sample_weight > 0)
    overall_lwlrap = sklearn.metrics.label_ranking_average_precision_score(
      truth[nonzero_weight_sample_indices, :] > 0,
      scores[nonzero_weight_sample_indices, :],
      sample_weight=sample_weight[nonzero_weight_sample_indices])
    return overall_lwlrap
```

In [42]:
```
print(pred_val_y.shape, y_val.shape)
print(np.sum(pred_val_y), np.sum(y_val))
# for ii in range(len(y_val)):
#     print(np.sum(pred_val_y[ii]), np.sum(y_val[ii]))
```

```
(994, 80) (994, 80)
-11160808.0 1128
```

In [43]:
```
print("lwlrap from sklearn.metrics for training data =", calculate_overall_l
wlrap_sklearn(y_trn, pred_train_y))
print("val lwlrap from sklearn.metrics =", calculate_overall_lwlrap_sklearn
(y_val, pred_val_y/10))

score, weight = calculate_per_class_lwlrap(y_val, pred_val_y)
lwlrap = (score * weight).sum()
print('direct calculation of val lwlrap : %.4f' % (lwlrap))
```

```
lwlrap from sklearn.metrics for training data = 0.9937980679367769
val lwlrap from sklearn.metrics = 0.7947953814827754
direct calculation of val lwlrap : 0.7948
```

Simple Error Analysis

In [44]:
```
idx = np.sum(y_val,axis=1) > 1
print(y_val[idx, :].shape, y_val[idx==False, :].shape)

print("val lwlrap for multi-labels =", calculate_overall_lwlrap_sklearn(y_va
l[idx], pred_val_y[idx]))
print("val lwlrap for single-label =", calculate_overall_lwlrap_sklearn(y_va
l[idx==False], pred_val_y[idx==False]))
```

```
(125, 80) (869, 80)
val lwlrap for multi-labels = 0.7788500531195031
val lwlrap for single-label = 0.7995477865991014
```

Predict Test Data with TTA

In [45]:
```

```

In [45]:
```
test_steps = np.ceil(float(len(x_test)) / float(BATCH_SIZE)).astype(int)
```

In [46]:
```
model.load_weights(checkpoint_file[0])

test_generator = FATTrainDataset.create_generator(
    x_test, x_test, BATCH_SIZE, (SIZE,SIZE,3), augument=False, shuffling=Fal
se, test_data=True)
pred_test_y = model.predict_generator(test_generator,steps=test_steps,verbos
e=1)

for kk in range(len(TTA)):
    for ii in range(TTA[kk]):
        test_generator = FATTrainDataset.create_generator(
        x_test, x_test, BATCH_SIZE, (SIZE,SIZE,3), augument=False, shufflin
g=False, test_data=True)

        pred_test_y += model.predict_generator(test_generator,steps=test_ste
ps,verbose=1)

    if kk+1 < len(TTA) and TTA[kk+1] > 0:
        model.load_weights(checkpoint_file[kk+1])
```

```
35/35 [==============================] - 1s 16ms/step
35/35 [==============================] - 1s 16ms/step
35/35 [==============================] - 1s 16ms/step
35/35 [==============================] - 1s 16ms/step
35/35 [==============================] - 1s 15ms/step
35/35 [==============================] - 1s 16ms/step
35/35 [==============================] - 1s 16ms/step
35/35 [==============================] - 1s 15ms/step
35/35 [==============================] - 1s 16ms/step
35/35 [==============================] - 1s 15ms/step
35/35 [==============================] - 1s 16ms/step
35/35 [==============================] - 1s 15ms/step
35/35 [==============================] - 1s 16ms/step
35/35 [==============================] - 1s 16ms/step
35/35 [==============================] - 1s 16ms/step
35/35 [==============================] - 1s 16ms/step
35/35 [==============================] - 1s 16ms/step
35/35 [==============================] - 1s 16ms/step
35/35 [==============================] - 1s 16ms/step
35/35 [==============================] - 1s 15ms/step
```

In [47]:
```
sort_idx = np.argsort(labels).astype(int)
```

In [48]:
```
print(sort_idx)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
 72 73 74 75 76 77 78 79]
```

In [49]:
```
sample_sub = pd.read_csv('../input/freesound-audio-tagging-2019/sample_submi
ssion.csv')
test_Y_sort = pred_test_y[:, sort_idx]
sample_sub.iloc[:, 1:] =  test_Y_sort
sample_sub.to_csv('submission.csv', index=False)

sample_sub.head()
```

Out[49]:

	fname	Accelerating_and_revving_and_vroom	Accordion	Acoustic_guitar	Applause	B
0	000ccb97.wav	-154.853333	-178.622345	-137.645279	-154.654419	-124.198
1	0012633b.wav	-62.905537	-155.926025	-138.813126	-119.286285	-128.623
2	001ed5f1.wav	-101.077438	-147.583435	-141.133286	-120.893265	-146.542
3	00294be0.wav	-173.398453	-147.612381	-148.860428	-205.831070	-120.307
4	003fde7a.wav	-167.963501	-154.409439	-134.879196	-167.252792	-130.279

Code to create FAT2019 Preprocessed Mel-spectrogram Dataset

This is the code to create [FAT2019 Preprocessed Mel-spectrogram Dataset (https://www.kaggle.com/daisukelab/fat2019_prep_mels1)](https://www.kaggle.com/daisukelab/fat2019_prep_mels1).

Creating noisy set is commented out due to kernel memory restriction. You can fully run in your local environment. No GPU used.

In [1]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from pathlib import Path
import matplotlib.pyplot as plt
from tqdm import tqdm_notebook
import IPython
import IPython.display
import PIL
import pickle

import torch
import torch.nn as nn
import torch.nn.functional as F

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will l
ist the files in the input directory

import os
print(os.listdir("../input"))

# Any results you write to the current directory are saved as output.
```

```
['fat2019_prep_mels1', 'freesound-audio-tagging-2019']
```

In [2]:
```
DATA = Path('../input/freesound-audio-tagging-2019')
#PREPROCESSED = Path('../input/fat2019_prep_mels1')
PREPROCESSED = Path('work/fat2019_prep_mels1')
WORK = Path('work')
Path(PREPROCESSED).mkdir(exist_ok=True, parents=True)
Path(WORK).mkdir(exist_ok=True, parents=True)

CSV_TRN_CURATED = DATA/'train_curated.csv'
CSV_TRN_NOISY = DATA/'train_noisy.csv'
CSV_SUBMISSION = DATA/'sample_submission.csv'

TRN_CURATED = DATA/'train_curated'
TRN_NOISY = DATA/'train_noisy'
TEST = DATA/'test'

MELS_TRN_CURATED = PREPROCESSED/'mels_train_curated.pkl'
MELS_TRN_NOISY = PREPROCESSED/'mels_train_noisy.pkl'
MELS_TEST = PREPROCESSED/'mels_test.pkl'

CSV_TRN_NOISY_BEST50S = PREPROCESSED/'trn_noisy_best50s.csv'
MELS_TRN_NOISY_BEST50S = PREPROCESSED/'mels_trn_noisy_best50s.pkl'

trn_curated_df = pd.read_csv(CSV_TRN_CURATED)
trn_noisy_df = pd.read_csv(CSV_TRN_NOISY)
test_df = pd.read_csv(CSV_SUBMISSION)
```

In [3]:
```
import librosa
import librosa.display
import random

from fastai import *
from fastai.callbacks import *
from fastai.vision import *
from fastai.vision.data import *


def read_audio(conf, pathname, trim_long_data):
    y, sr = librosa.load(pathname, sr=conf.sampling_rate)
    # trim silence
    if 0 < len(y): # workaround: 0 length causes error
        y, _ = librosa.effects.trim(y) # trim, top_db=default(60)
    # make it unified length to conf.samples
    if len(y) > conf.samples: # long enough
        if trim_long_data:
            y = y[0:0+conf.samples]
    else: # pad blank
        padding = conf.samples - len(y)    # add padding at both ends
        offset = padding // 2
        y = np.pad(y, (offset, conf.samples - len(y) - offset), conf.padmod
e)
    return y


def audio_to_melspectrogram(conf, audio):
    spectrogram = librosa.feature.melspectrogram(audio,
                                                 sr=conf.sampling_rate,
                                                 n_mels=conf.n_mels,
                                                 hop_length=conf.hop_length,
                                                 n_fft=conf.n_fft,
                                                 fmin=conf.fmin,
                                                 fmax=conf.fmax)
    spectrogram = librosa.power_to_db(spectrogram)
    spectrogram = spectrogram.astype(np.float32)
    return spectrogram


def show_melspectrogram(conf, mels, title='Log-frequency power spectrogram
'):
    librosa.display.specshow(mels, x_axis='time', y_axis='mel',
                             sr=conf.sampling_rate, hop_length=conf.hop_leng
th,
                            fmin=conf.fmin, fmax=conf.fmax)
    plt.colorbar(format='%+2.0f dB')
    plt.title(title)
    plt.show()


def read_as_melspectrogram(conf, pathname, trim_long_data, debug_display=Fal
se):
    x = read_audio(conf, pathname, trim_long_data)
    mels = audio_to_melspectrogram(conf, x)
    if debug_display:
        IPython.display.display(IPython.display.Audio(x, rate=conf.sampling_
rate))
        show_melspectrogram(conf, mels)
    return mels


class conf:
    sampling_rate = 44100
    duration = 2 # sec
    hop_length = 347*duration # to make time steps 128
    fmin = 20
    fmax = sampling_rate // 2
```

In [4]:
```
def mono_to_color(X, mean=None, std=None, norm_max=None, norm_min=None, eps=
1e-6):
    # Stack X as [X,X,X]
    X = np.stack([X, X, X], axis=-1)

    # Standardize
    mean = mean or X.mean()
    X = X - mean
    std = std or X.std()
    Xstd = X / (std + eps)
    _min, _max = Xstd.min(), Xstd.max()
    norm_max = norm_max or _max
    norm_min = norm_min or _min
    if (_max - _min) > eps:
        # Normalize to [0, 255]
        V = Xstd
        V[V < norm_min] = norm_min
        V[V > norm_max] = norm_max
        V = 255 * (V - norm_min) / (norm_max - norm_min)
        V = V.astype(np.uint8)
    else:
        # Just zero
        V = np.zeros_like(Xstd, dtype=np.uint8)
    return V


def convert_wav_to_image(df, source):
    X = []
    for i, row in tqdm_notebook(df.iterrows()):
        x = read_as_melspectrogram(conf, source/str(row.fname), trim_long_da
ta=False)
        x_color = mono_to_color(x)
        X.append(x_color)
    return X


def save_as_pkl_binary(obj, filename):
    """Save object as pickle binary file.
    Thanks to https://stackoverflow.com/questions/19201290/how-to-save-a-dic
tionary-to-a-file/32216025
    """
    with open(filename, 'wb') as f:
        pickle.dump(obj, f, pickle.HIGHEST_PROTOCOL)


def load_pkl(filename):
    """Load pickle object from file."""
    with open(filename, 'rb') as f:
        return pickle.load(f)
```

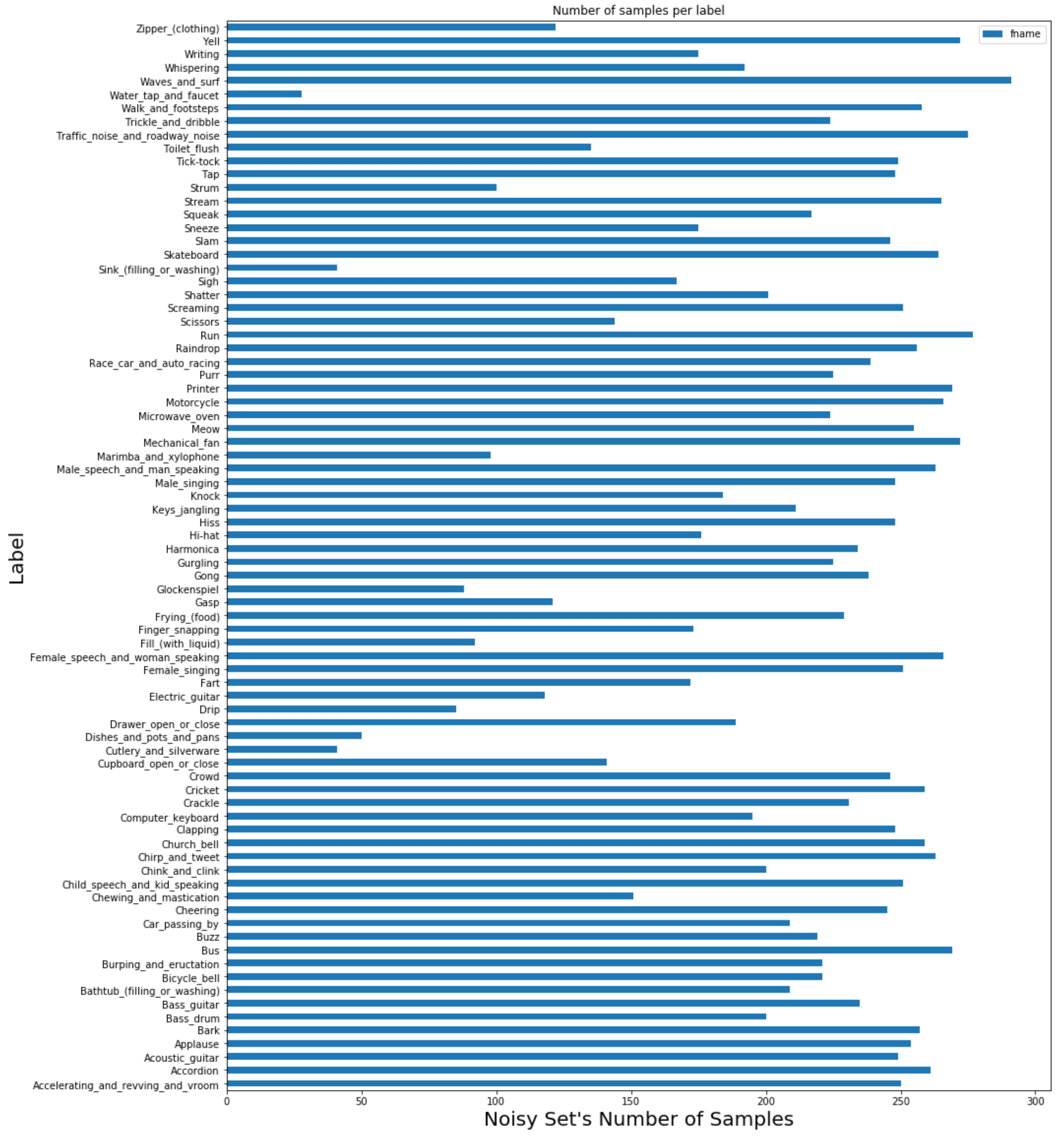
In [5]:
```
conf = get_default_conf()

def convert_dataset(df, source_folder, filename):
    X = convert_wav_to_image(df, source=source_folder)
    save_as_pkl_binary(X, filename)
    print(f'Created {filename}')
    return X

convert_dataset(trn_curated_df, TRN_CURATED, MELS_TRN_CURATED);
convert_dataset(test_df, TEST, MELS_TEST);
```

```
Created work/fat2019_prep_mels1/mels_train_curated.pkl
```

```
Created work/fat2019_prep_mels1/mels_test.pkl
```

Creating Best 50s

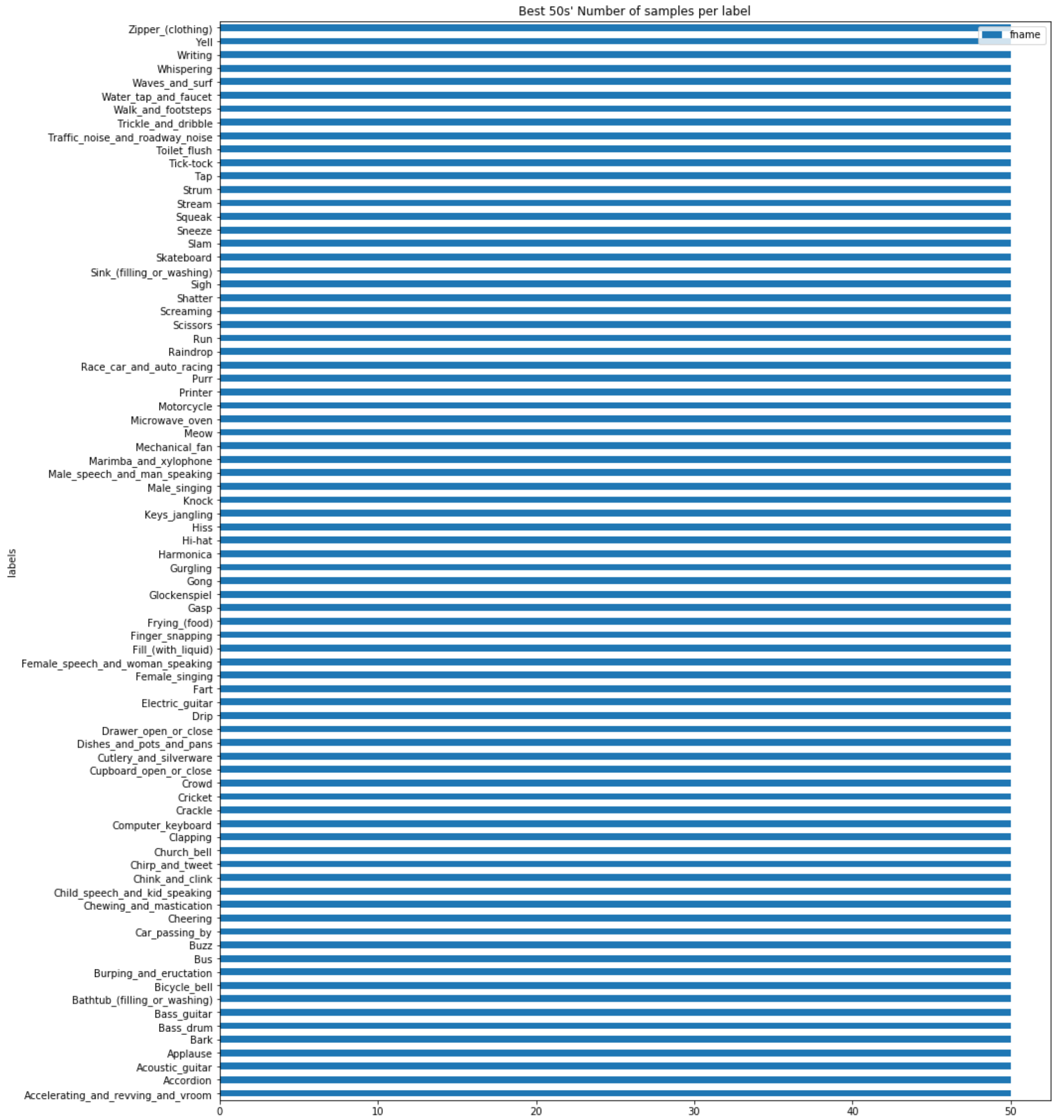
In [6]:
```
df = trn_noisy_df.copy()
df['singled'] = ~df.labels.str.contains(',')
singles_df = df[df.singled]

cat_gp = (singles_df.groupby(
    ['labels']).agg({
    'fname':'count'
}).reset_index()).set_index('labels')

plot = cat_gp.plot(
    kind='barh',
    title="Number of samples per label",
    figsize=(15,20))
plot.set_xlabel("Noisy Set's Number of Samples", fontsize=20)
plot.set_ylabel("Label", fontsize=20);
```



In [7]:
```
labels = singles_df.labels.unique()
labels, len(labels)
```

Out[7]:
```
(array(['Bathtub_(filling_or_washing)', 'Motorcycle', 'Raindrop', 'Bass_guita
r', ..., 'Glockenspiel',
        'Dishes_and_pots_and_pans', 'Sink_(filling_or_washing)', 'Water_tap_a
nd_faucet'], dtype=object),
 80)
```

In [8]:
```
idxes_best50s = np.array([random.choices(singles_df[(singles_df.labels ==
l)].index, k=50)
                            for l in labels]).ravel()
best50s_df = singles_df.loc[idxes_best50s]

grp = (best50s_df.groupby(
    ['labels']).agg({
    'fname':'count'
}).reset_index()).set_index('labels')
grp.plot( kind='barh', title="Best 50s' Number of samples per label", figsiz
e=(15,20));
```



In [9]:
```
best50s_df.to_csv(CSV_TRN_NOISY_BEST50S, index=False)
```

Now best 50s are selected

Making preprocessed data is as follows, but you have to run locally. Kernel cannot hold all the noisy preprocessed data on memory.

In [10]:
```
# Convert noisy set first
#X_trn_noisy = convert_dataset(trn_noisy_df, TRN_NOISY, MELS_TRN_NOISY)

# Then choose preprocessed data for 50s, and save it
#X = [X_trn_noisy[i] for i in idxes_best50s]
#save_as_pkl_binary(X, MELS_TRN_NOISY_BEST50S)
```

Birectional LSTM model for audio labeling with Keras

In this Kaggle kernel we will use the curated data from the "Freesound Audio Tagging 2019" competition to predict the labels of .wav files.

Table of contents

Data Description

From Kaggle's data page (https://www.kaggle.com/c/freesound-audio-tagging-2019/data) for the competition:

The curated subset is a small set of manually-labeled data from FSD.

Number of clips/class: 75 except in a few cases (where there are less)

Total number of clips: 4970

Avge number of labels/clip: 1.2

Total duration: 10.5 hours

The duration of the audio clips ranges from 0.3 to 30s due to the diversity of the sound categories and the preferences of Freesound users when recording/uploading sounds. It can happen that a few of these audio clips present additional acoustic material beyond the provided ground truth label(s).

Test Set:

The test set is used for system evaluation and consists of manually-labeled data from FSD. Since most of the train data come from YFCC, some acoustic domain mismatch between the train and test set can be expected. All the acoustic material present in the test set is labeled, except human error, considering the vocabulary of 80 classes used in the competition.

Columns:

fname: the audio file name, eg, 0006ae4e.wav *labels*: the audio classification label(s) (ground truth). Note that the number of labels per clip can be one, eg, Bark or more, eg, "Walk_and_footsteps,Slam".

Dependencies

In [1]:
```
# Dependencies
import numpy as np
import pandas as pd
import os
import librosa
import matplotlib.pyplot as plt
import gc
import time
from tqdm import tqdm, tqdm_notebook; tqdm.pandas() # Progress bar
from sklearn.metrics import label_ranking_average_precision_score
from sklearn.model_selection import train_test_split

# Machine Learning
import tensorflow as tf
from keras import backend as K
from keras.engine.topology import Layer
from keras import initializers, regularizers, constraints, optimizers, layer
s
from keras.layers import (Dense, Bidirectional, CuDNNLSTM, ELU,
                          Dropout, LeakyReLU, Conv1D, BatchNormalization)
from keras.models import Sequential
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping

# Path specifications
KAGGLE_DIR = '../input/'
train_curated_path = KAGGLE_DIR + 'train_curated/'
test_path = KAGGLE_DIR + 'test/'

# Set seed for reproducability
seed = 1234
np.random.seed(seed)
tf.set_random_seed(seed)

# File sizes and specifications
print('\n# Files and file sizes')
for file in os.listdir(KAGGLE_DIR):
    print('{}| {} MB'.format(file.ljust(30),
                             str(round(os.path.getsize(KAGGLE_DIR + file) /
1000000, 2))))

# For keeping time. GPU limit for this competition is set to 60 min.
t_start = time.time()
```

```
# Files and file sizes
train_curated.csv              | 0.14 MB
train_noisy.csv                | 0.58 MB
test                           | 0.04 MB
sample_submission.csv          | 0.19 MB
train_curated                  | 0.14 MB
train_noisy                    | 0.55 MB

Using TensorFlow backend.
```

Evaluation metric

From the competition evaluation page (https://www.kaggle.com/c/freesound-audio-tagging-2019/overview/evaluation):

The task consists of predicting the audio labels (tags) for every test clip. Some test clips bear one label while others bear several labels. The predictions are to be done at the clip level, i.e., no start/end timestamps for the sound events are required.

The primary competition metric will be label-weighted label-ranking average precision (https://scikit-learn.org/stable/modules/model_evaluation.html#label-ranking-average-precision) (lwlrap, pronounced "Lol wrap"). This measures the average precision of retrieving a ranked list of relevant labels for each test clip (i.e., the system ranks all the available labels, then the precisions of the ranked lists down to each true label are averaged). This is a generalization of the mean reciprocal rank measure (used in last year's edition of the competition) for the case where there can be multiple true labels per test item. The novel "label-weighted" part means that the overall score is the average over all the labels in the test set, where each label receives equal weight (by contrast, plain lrap gives each test item equal weight, thereby discounting the contribution of individual labels when they appear on the same item as multiple other labels).

The formula for label-ranking average precision (LRAP) is as follows:

$$LRAP(y, \hat{f}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} \frac{1}{||y_i||_0} \sum_{j:y_{ij}=1} \frac{|\mathcal{L}_{ij}|}{\text{rank}_{ij}}$$

Happily, the evaluation metric is provided by Kaggle and can be found in this Google Colab file (https://colab.research.google.com/drive/1AgPdhSp7ttY18O3fEoHOQKlt_3HJDLi8#scrollTo=52LPXQNPppex).

In [2]:
```
def calculate_overall_lwlrap_sklearn(truth, scores):
    """Calculate the overall lwlrap using sklearn.metrics.lrap."""
    # sklearn doesn't correctly apply weighting to samples with no labels, s
o just skip them.
    sample_weight = np.sum(truth > 0, axis=1)
    nonzero_weight_sample_indices = np.flatnonzero(sample_weight > 0)
    overall_lwlrap = label_ranking_average_precision_score(
        truth[nonzero_weight_sample_indices, :] > 0,
        scores[nonzero_weight_sample_indices, :],
        sample_weight=sample_weight[nonzero_weight_sample_indices])
    return overall_lwlrap
```

Helper Functions and Preprocessing

I got the inspiration for most of the preprocessing steps and the attention layer from this Kaggle kernel (https://www.kaggle.com/chewzy/gru-w-attention-baseline-model-curated).

In [3]:
```
n_classes = 80

def split_and_label(rows_labels, n_classes):
    '''
    Retrieves a list of all the relevant classes. This is necessary due to
    the multi-labeling of the initial csv file.
    '''
    row_labels_list = []
    for row in rows_labels:
        row_labels = row.split(',')
        labels_array = np.zeros((n_classes))
        for label in row_labels:
            index = label_mapping[label]
            labels_array[index] = 1
        row_labels_list.append(labels_array)
    return row_labels_list
```

In [4]:
```
# Load in data
df = pd.read_csv(KAGGLE_DIR + 'train_curated.csv')
test_df = pd.read_csv(KAGGLE_DIR + 'sample_submission.csv')

# Retrieve labels
label_columns = test_df.columns[1:]
label_mapping = dict((label, index) for index, label in enumerate(label_colu
mns))
for col in label_columns:
    df[col] = 0
df[label_columns] = split_and_label(df['labels'], n_classes)
df['num_labels'] = df[label_columns].sum(axis=1)
```

In [5]:
```
# Check dataframes
print('Training dataframe:')
display(df.head(3))
print('Testing dataframe:')
test_df.head(3)
```

```
Training dataframe:
```

	fname	labels	Accelerating_and_revving_and_vroom	Accordion	Acoustic_guitar	Appla
0	0006ae4e.wav	Bark	0.0	0.0	0.0	
1	0019ef41.wav	Raindrop	0.0	0.0	0.0	
2	001ec0ad.wav	Finger_snapping	0.0	0.0	0.0	

```
Testing dataframe:
```

Out[5]:

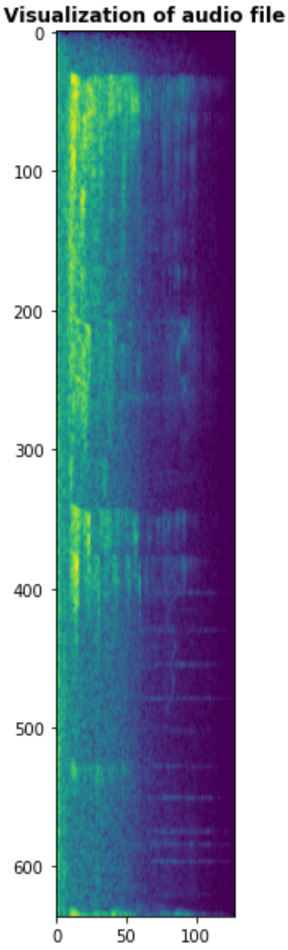
	fname	Accelerating_and_revving_and_vroom	Accordion	Acoustic_guitar	Applause	Bark	Bass_
0	000ccb97.wav	0	0	0	0	0	
1	0012633b.wav	0	0	0	0	0	
2	001ed5f1.wav	0	0	0	0	0	

In [6]:
```
# Preprocessing parameters
sr = 44100 # Sampling rate
duration = 5
hop_length = 347 # to make time steps 128
fmin = 20
fmax = sr // 2
n_mels = 128
n_fft = n_mels * 20
samples = sr * duration
```

In [7]:
```
def read_audio(path):
    '''
    Reads in the audio file and returns
    an array that we can turn into a melspectogram
    '''
    y, _ = librosa.core.load(path, sr=44100)
    # trim silence
    if 0 < len(y): # workaround: 0 length causes error
        y, _ = librosa.effects.trim(y)
    if len(y) > samples: # long enough
        y = y[0:0+samples]
    else: # pad blank
        padding = samples - len(y)
        offset = padding // 2
        y = np.pad(y, (offset, samples - len(y) - offset), 'constant')
    return y

def audio_to_melspectrogram(audio):
    '''
    Convert to melspectrogram after audio is read in
    '''
    spectrogram = librosa.feature.melspectrogram(audio,
                                                 sr=sr,
                                                 n_mels=n_mels,
                                                 hop_length=hop_length,
                                                 n_fft=n_fft,
                                                 fmin=fmin,
                                                 fmax=fmax)
    return librosa.power_to_db(spectrogram).astype(np.float32)

def read_as_melspectrogram(path):
    '''
    Convert audio into a melspectrogram
    so we can use machine learning
    '''
    mels = audio_to_melspectrogram(read_audio(path))
    return mels

def convert_wav_to_image(df, path):
    X = []
    for _,row in tqdm_notebook(df.iterrows()):
        x = read_as_melspectrogram('{}/{}'.format(path[0],
                                                  str(row['fname'])))
        X.append(x.transpose())
    return X

def normalize(img):
    '''
    Normalizes an array
    (subtract mean and divide by standard deviation)
    '''
    eps = 0.001
    if np.std(img) != 0:
        img = (img - np.mean(img)) / np.std(img)
    else:
        img = (img - np.mean(img)) / eps
    return img

def normalize_dataset(X):
    '''
    Normalizes list of arrays
    (subtract mean and divide by standard deviation)
    '''
    normalized_dataset = []
    for img in X:
        normalized = normalize(img)
        normalized_dataset.append(normalized)
    return normalized_dataset
```

In [8]:
```
# Preprocess dataset and create validation sets
X = np.array(convert_wav_to_image(df, [train_curated_path]))
X = normalize_dataset(X)
Y = df[label_columns].values
x_train, x_val, y_train, y_val = train_test_split(X, Y, test_size=0.1, rando
m_state=seed)
```

In [9]:
```
# Visualize an melspectogram example
plt.figure(figsize=(15,10))
plt.title('Visualization of audio file', weight='bold')
plt.imshow(X[0]);
```



Modeling

My main inspiration for this architecture has been [this paper (https://arxiv.org/pdf/1602.05875v3.pdf)](https://arxiv.org/pdf/1602.05875v3.pdf).

In [10]:
```
class Attention(Layer):
    def __init__(self, step_dim,
                 W_regularizer=None, b_regularizer=None,
                 W_constraint=None, b_constraint=None,
                 bias=True, **kwargs):
        self.supports_masking = True
        self.init = initializers.get('glorot_uniform')
        self.W_regularizer = regularizers.get(W_regularizer)
        self.b_regularizer = regularizers.get(b_regularizer)
        self.W_constraint = constraints.get(W_constraint)
        self.b_constraint = constraints.get(b_constraint)
        self.bias = bias
        self.step_dim = step_dim
        self.features_dim = 0
        super(Attention, self).__init__(**kwargs)

    def build(self, input_shape):
        assert len(input_shape) == 3

        self.W = self.add_weight((input_shape[-1],),
                                 initializer=self.init,
                                 name='{}_W'.format(self.name),
                                 regularizer=self.W_regularizer,
                                 constraint=self.W_constraint)
        self.features_dim = input_shape[-1]

        if self.bias:
            self.b = self.add_weight((input_shape[1],),
                                     initializer='zero',
                                     name='{}_b'.format(self.name),
                                     regularizer=self.b_regularizer,
                                     constraint=self.b_constraint)
        else:
            self.b = None
        self.built = True

    def compute_mask(self, input, input_mask=None):
        return None

    def call(self, x, mask=None):
        features_dim = self.features_dim
        step_dim = self.step_dim

        eij = K.reshape(K.dot(K.reshape(x, (-1, features_dim)),
                        K.reshape(self.W, (features_dim, 1))), (-1, step_di
m))
        if self.bias:
            eij += self.b
        eij = K.tanh(eij)
        a = K.exp(eij)
        if mask is not None:
            a *= K.cast(mask, K.floatx())
        a /= K.cast(K.sum(a, axis=1, keepdims=True) + K.epsilon(), K.floatx
())
        a = K.expand_dims(a)
        weighted_input = x * a
        return K.sum(weighted_input, axis=1)

    def compute_output_shape(self, input_shape):
        return input_shape[0],  self.features_dim
```

In [11]:
```
# Neural network model
input_shape = (636,128)
optimizer = Adam(0.005, beta_1=0.1, beta_2=0.001, amsgrad=True)
n_classes = 80

model = Sequential()
model.add(Bidirectional(CuDNNLSTM(256, return_sequences=True), input_shape=i
nput_shape))
model.add(Attention(636))
model.add(Dropout(0.2))
model.add(Dense(400))
model.add(ELU())
model.add(Dropout(0.2))
model.add(Dense(n_classes, activation='softmax'))
model.compile(loss='categorical_crossentropy',
              optimizer=optimizer,
              metrics=['acc'])
```

```
WARNING:tensorflow:From /opt/conda/lib/python3.6/site-packages/tensorflow/pyt
hon/framework/op_def_library.py:263: colocate_with (from tensorflow.python.fr
amework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From /opt/conda/lib/python3.6/site-packages/keras/backend/
tensorflow_backend.py:3445: calling dropout (from tensorflow.python.ops.nn_op
s) with keep_prob is deprecated and will be removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - k
eep_prob`.
```

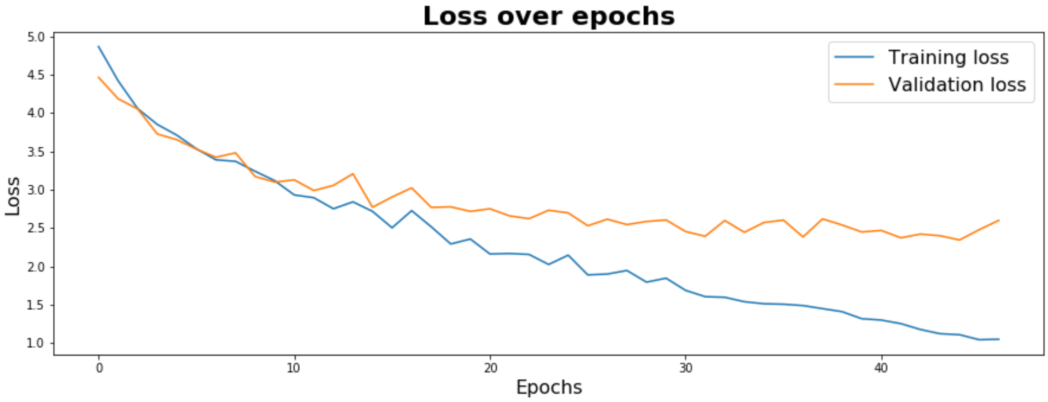
In [12]:
```
# Train model
es = EarlyStopping(monitor='val_acc', mode='max', verbose=1, patience=10)
hist = model.fit(np.array(x_train),
          y_train,
          batch_size=1024,
          epochs=500,
          validation_data=(np.array(x_val), y_val),
          callbacks = [es])
```

```
WARNING:tensorflow:From /opt/conda/lib/python3.6/site-packages/tensorflow/pyt
hon/ops/math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is d
eprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train on 4473 samples, validate on 497 samples
Epoch 1/500
4473/4473 [==============================] - 12s 3ms/step - loss: 4.8638 - ac
c: 0.0483 - val_loss: 4.4624 - val_acc: 0.0986
Epoch 2/500
4473/4473 [==============================] - 6s 1ms/step - loss: 4.4178 - ac
c: 0.1008 - val_loss: 4.1836 - val_acc: 0.1167
Epoch 3/500
4473/4473 [==============================] - 6s 1ms/step - loss: 4.0545 - ac
c: 0.1567 - val_loss: 4.0502 - val_acc: 0.1529
Epoch 4/500
4473/4473 [==============================] - 6s 1ms/step - loss: 3.8491 - ac
c: 0.1827 - val_loss: 3.7245 - val_acc: 0.1811
Epoch 5/500
4473/4473 [==============================] - 6s 1ms/step - loss: 3.7089 - ac
c: 0.2077 - val_loss: 3.6501 - val_acc: 0.1972
Epoch 6/500
4473/4473 [==============================] - 6s 1ms/step - loss: 3.5312 - ac
c: 0.2397 - val_loss: 3.5270 - val_acc: 0.2274
Epoch 7/500
4473/4473 [==============================] - 6s 1ms/step - loss: 3.3873 - ac
c: 0.2676 - val_loss: 3.4207 - val_acc: 0.2314
Epoch 8/500
4473/4473 [==============================] - 6s 1ms/step - loss: 3.3682 - ac
c: 0.2707 - val_loss: 3.4783 - val_acc: 0.2455
Epoch 9/500
4473/4473 [==============================] - 6s 1ms/step - loss: 3.2380 - ac
c: 0.2862 - val_loss: 3.1699 - val_acc: 0.3058
Epoch 10/500
4473/4473 [==============================] - 6s 1ms/step - loss: 3.1157 - ac
c: 0.3163 - val_loss: 3.0972 - val_acc: 0.2978
Epoch 11/500
4473/4473 [==============================] - 6s 1ms/step - loss: 2.9296 - ac
c: 0.3588 - val_loss: 3.1261 - val_acc: 0.2817
Epoch 12/500
4473/4473 [==============================] - 6s 1ms/step - loss: 2.8936 - ac
c: 0.3541 - val_loss: 2.9871 - val_acc: 0.3099
Epoch 13/500
4473/4473 [==============================] - 6s 1ms/step - loss: 2.7495 - ac
c: 0.3906 - val_loss: 3.0536 - val_acc: 0.3038
Epoch 14/500
4473/4473 [==============================] - 6s 1ms/step - loss: 2.8394 - ac
c: 0.3651 - val_loss: 3.2061 - val_acc: 0.2857
Epoch 15/500
4473/4473 [==============================] - 6s 1ms/step - loss: 2.7153 - ac
c: 0.3937 - val_loss: 2.7690 - val_acc: 0.3763
Epoch 16/500
4473/4473 [==============================] - 6s 1ms/step - loss: 2.5014 - ac
c: 0.4420 - val_loss: 2.9024 - val_acc: 0.3461
Epoch 17/500
4473/4473 [==============================] - 6s 1ms/step - loss: 2.7248 - ac
c: 0.3957 - val_loss: 3.0215 - val_acc: 0.3501
Epoch 18/500
4473/4473 [==============================] - 6s 1ms/step - loss: 2.5144 - ac
c: 0.4384 - val_loss: 2.7673 - val_acc: 0.3682
Epoch 19/500
4473/4473 [==============================] - 6s 1ms/step - loss: 2.2901 - ac
c: 0.4851 - val_loss: 2.7746 - val_acc: 0.3964
Epoch 20/500
4473/4473 [==============================] - 6s 1ms/step - loss: 2.3551 - ac
c: 0.4681 - val_loss: 2.7149 - val_acc: 0.4145
Epoch 21/500
4473/4473 [==============================] - 6s 1ms/step - loss: 2.1597 - ac
```

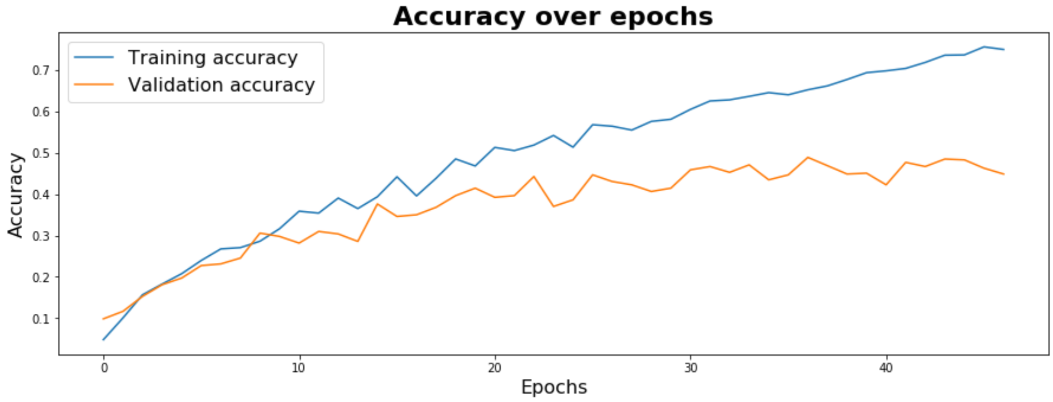
Visualization and Evaluation

Simple visualizations to keep track of the loss and accuracy over the epochs.

In [13]:
```
# Visualize loss
loss = hist.history['loss']
val_loss = hist.history['val_loss']
stopped_epoch = es.stopped_epoch
epochs = range(stopped_epoch+1)

plt.figure(figsize=(15,5))
plt.plot(epochs, loss)
plt.plot(epochs, val_loss)
plt.title('Loss over epochs', weight='bold', fontsize=22)
plt.xlabel('Epochs', fontsize=16)
plt.ylabel('Loss', fontsize=16)
plt.legend(['Training loss', 'Validation loss'], fontsize=16)
plt.show()
```



In [14]:
```
# Visualize Accuracy
acc = hist.history['acc']
val_acc = hist.history['val_acc']
epochs = range(stopped_epoch+1)

plt.figure(figsize=(15,5))
plt.plot(epochs, acc)
plt.plot(epochs, val_acc)
plt.title('Accuracy over epochs', weight='bold', fontsize=22)
plt.xlabel('Epochs', fontsize=16)
plt.ylabel('Accuracy', fontsize=16)
plt.legend(['Training accuracy', 'Validation accuracy'], fontsize=16)
plt.show()
```



Training accuracy LWLRAP score:

In [15]:
```
# Make predictions for training set and validation set
y_train_pred = model.predict(np.array(x_train))
y_val_pred = model.predict(np.array(x_val))
train_lwlrap = calculate_overall_lwlrap_sklearn(y_train, y_train_pred)
val_lwlrap = calculate_overall_lwlrap_sklearn(y_val, y_val_pred)

# Check training and validation LWLRAP score
print('Training LWLRAP : {}'.format(round(train_lwlrap,4)))
print('Validation LWLRAP : {}'.format(round(val_lwlrap,4)))
```

```
Training LWLRAP : 0.8752
Validation LWLRAP : 0.6121
```

Predictions and submission

Preprocess the test set, make predictions and store them as a csv file for our submission.

In [16]:
```
# Prepare test set
X_test = np.array(convert_wav_to_image(test_df, [test_path]))
X_test = normalize_dataset(X_test)
# Make predictions
predictions = model.predict(np.array(X_test))
# Save predictions in a csv file
test_df[label_columns] = predictions
test_df.to_csv('submission.csv', index=False)
```

Final checks

Lastly, we check if the submission format is correct and if we are under the one hour limit of GPU time.

In [17]:
```
# Check submission format
display(test_df.head())

# Check if we are under one hour of GPU time
t_finish = time.time()
total_time = round((t_finish-t_start)/3600, 4)
print('Kernel runtime = {} hours ({} minutes)'.format(total_time,
                                                      int(total_time*60)))
```

	fname	Accelerating_and_revving_and_vroom	Accordion	Acoustic_guitar	Applause	Bark
0	000ccb97.wav	0.000161	0.000009	7.580730e-05	2.552045e-04	0.001516
1	0012633b.wav	0.091847	0.000155	8.812740e-05	5.745049e-05	0.001546
2	001ed5f1.wav	0.000046	0.000007	1.403562e-05	5.386537e-05	0.000021
3	00294be0.wav	0.000003	0.000001	8.072470e-07	7.080239e-08	0.000127
4	003fde7a.wav	0.000021	0.000233	3.527369e-06	4.100578e-06	0.000005

```
Kernel runtime = 0.2056 hours (12 minutes)
```

If you like this Kaggle kernel, feel free to give an upvote and leave a comment! I will try to implement your suggestions in this kernel!

To classify audio, you first need present it somehow to the classifier. You may notice everyone is talking about **spectrogram, FFT, STFT, MFCC**, but why don't we **just use audio?** What does it all stand for?

Here comes a little explanation!

tip: most interesting things are marked as **QUESTION**

In [1]:
```
import os
from os.path import isdir, join
from pathlib import Path
import pandas as pd

# Math
import numpy as np
from scipy.fftpack import fft
from scipy import signal
from scipy.io import wavfile
import librosa

from sklearn.decomposition import PCA

# Visualization
import matplotlib.pyplot as plt
import seaborn as sns
import IPython.display as ipd
import librosa.display

import plotly.offline as py
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
import plotly.tools as tls
import pandas as pd

%matplotlib inline
```

Samples

We can load the first two samples we'll be working on

In [2]:
```
train_audio_path = '../input/train_curated/'
# 8a8110c2 c2aff189 d7d25898 0a2895b8 6459fc05 54940c5c 024e0fbe c6f8f09e f4
6cc65b
# 1acaf122 a0a85eae da3a5cd5 412c28dd 0f301184 2ce5262c
sample_rate, samples1 = wavfile.read(os.path.join(train_audio_path, '98b0df7
6.wav'))
sample_rate, samples2 = wavfile.read(os.path.join(train_audio_path, 'd7d2589
8.wav'))
```

How do they sound?

In [3]:
```
ipd.Audio(samples1, rate=sample_rate)
```

Out[3]:

0:00:00 / 13:31:36

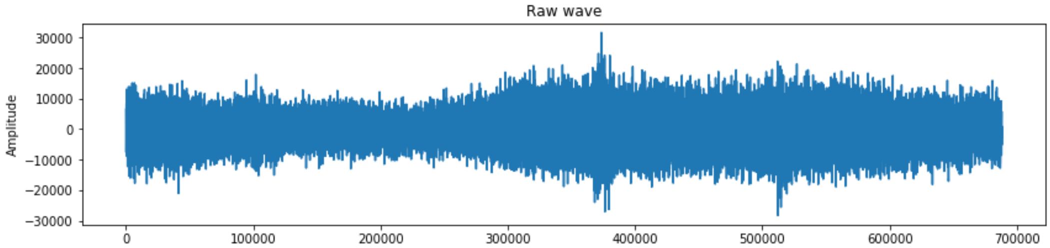
In [4]:
```
ipd.Audio(samples2, rate=sample_rate)
```

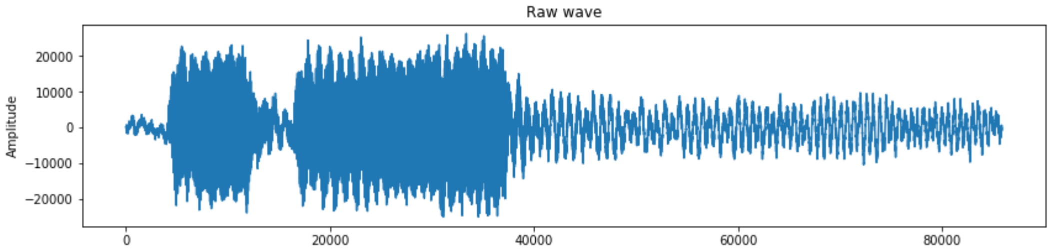
Out[4]:

0:00:00 / 13:31:36

In [5]:
```
def plot_raw_wave(samples):
    plt.figure(figsize=(14, 3))
    plt.title('Raw wave')
    plt.ylabel('Amplitude')
    # ax1.plot(np.linspace(0, sample_rate/len(samples1), sample_rate), sampl
es1)
    plt.plot(samples)
    plt.show()
```

In [6]:
```
plot_raw_wave(samples1)
plot_raw_wave(samples2)
```





You can easily SEE 2 beeps in the second wave, and constant noise in first one. The problem is that it is impossible to understand or interpret the pitch of a sound watching physical illustration as above.

The first thing we need to understand: the sample rate. yYou can read about it here (https://en.wikipedia.org/wiki/Sampling_(signal_processing))

Sampling rate: Intuition

So wait, does it mean one second of a recording has 44100 samples ('features'), so the longest recordings will have *30*44100 = 1323000* elements? Yes. That's one of the reasons why we need some different representation of an audio signal.

But hey, if this fluctuations are what directly come into the human ear, shouldn't we use it directly? Not really. Human hearing is a tough topic, not well understood, but we can assume that our brain hears rather something like frequencies (Reference (https://en.wikipedia.org/wiki/Place_theory_(hearing)))

What is frequency?

An explanation here (http://www.indiana.edu/~emusic/etext/acoustics/chapter1_frequency.shtml) and a nice tutorial about calculating the frequencies here: link (https://www.youtube.com/watch?v=r18Gi8lSkfM). The mathematical tool for that is Fast Fourier Transform (FFT)

Briefly:

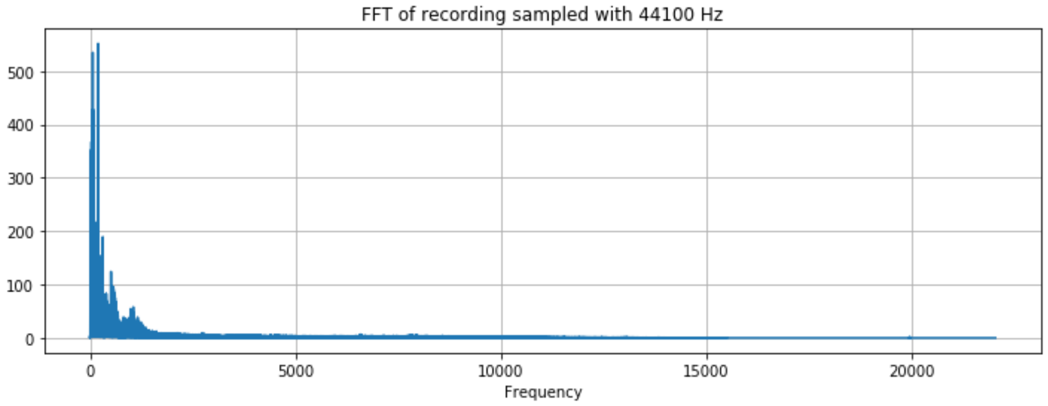
- low sounds are long waves
- higher sounds are shorter waves

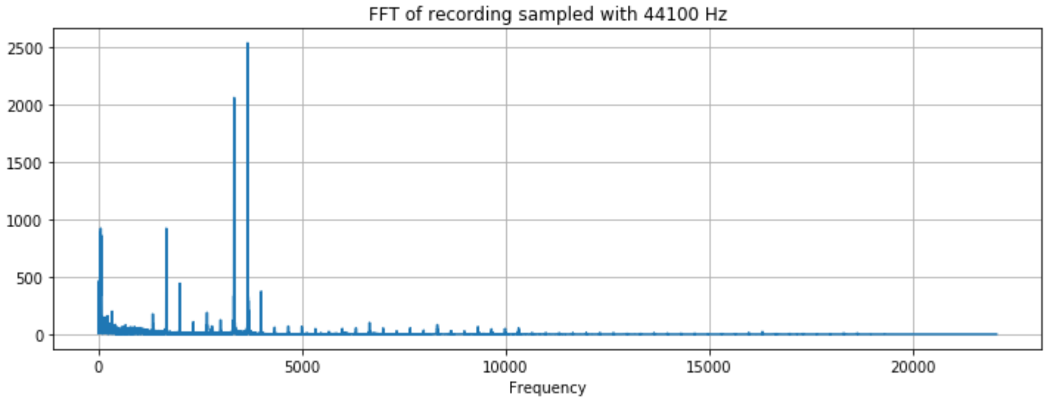
To calculate frequencies, we calculate the amount of long waves and the amount of short waves

We analyze our two examples: the one with "low" sounds, and one with high pitches.

In [7]:
```
def custom_fft(y, fs):
    T = 1.0 / fs
    N = y.shape[0]
    yf = fft(y)
    xf = np.linspace(0.0, 1.0/(2.0*T), N//2)
    vals = 2.0/N * np.abs(yf[0:N//2])  # FFT is simmetrical, so we take just
the first half
    # FFT is also complex, to we take just the real part (abs)
    return xf, vals

def plot_custom_fft(samples, sample_rate):
    xf, vals = custom_fft(samples, sample_rate)
    plt.figure(figsize=(12, 4))
    plt.title('FFT of recording sampled with ' + str(sample_rate) + ' Hz')
    plt.plot(xf, vals)
    plt.xlabel('Frequency')
    plt.grid()
    plt.show()
```

```
In [8]: plot_custom_fft(samples1, sample_rate)
        plot_custom_fft(samples2, sample_rate)
```





Great to see that FFT really shows bigger amplitude in low freqs in machine noise and big amplitude in high freqs for beeps.

FFT is not enough

We can indeed calculate all the frequencies in audio. But sound differs in time, so maybe we should calculate the frequencies for a small part of a signal, to show the time dependencies?

We can easily do that. We call the result **spectrogram**

How to create spectrogram in 3 steps:

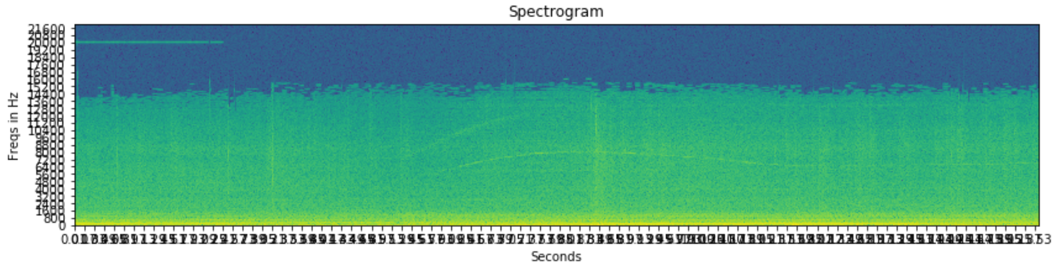
1. Cut the signal by overlapping windows 20 ms windows
2. Find short and long waves (frequencies in this window) using FFT
3. Concatenate the calculated frequencies.

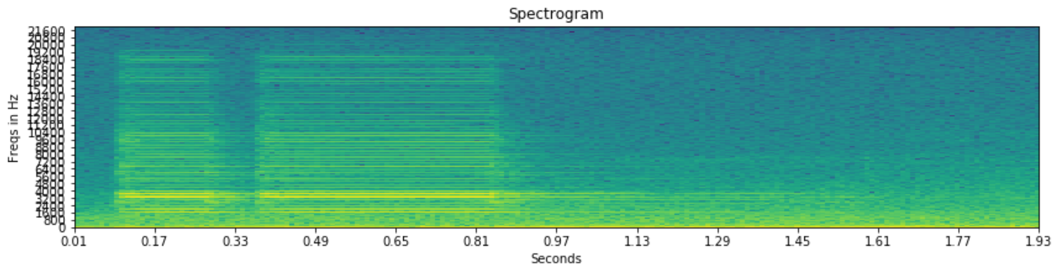
Let's implement it:

In [9]:
```
def log_specgram(audio, sample_rate, window_size=20,
                 step_size=10, eps=1e-10):
    nperseg = int(round(window_size * sample_rate / 1e3))
    noverlap = int(round(step_size * sample_rate / 1e3))
    freqs, times, spec = signal.spectrogram(audio,
                                    fs=sample_rate,
                                    window='hann',
                                    nperseg=nperseg,
                                    noverlap=noverlap,
                                    detrend=False)
    return freqs, times, np.log(spec.T.astype(np.float32) + eps)

def plot_log_specgram(audio, sample_rate, window_size=20, step_size=10, eps=
1e-10):

    fig = plt.figure(figsize=(14, 3))
    freqs, times, spectrogram = log_specgram(audio, sample_rate)
    plt.imshow(spectrogram.T, aspect='auto', origin='lower',
               extent=[times.min(), times.max(), freqs.min(), freqs.max()])
    plt.yticks(freqs[::16])
    plt.xticks(times[::16])
    plt.title('Spectrogram')
    plt.ylabel('Freqs in Hz')
    plt.xlabel('Seconds')
    plt.show()
```

And now we visualize our two examples using the spectrogram

In [10]:
```
plot_log_specgram(samples1, sample_rate)
plot_log_specgram(samples2, sample_rate)
```





Important:

This is really a representation that **reflects and takes into account the audible properties of a signal**. Do you agree?

Nyquist theorem

Interesting property: according to [Nyquist theorem (https://en.wikipedia.org/wiki/Nyquist_rate)](https://en.wikipedia.org/wiki/Nyquist_rate) we can hear only frequencies twice lower than the sampling rate.

I not going to present this fact here, but what I want to say is:

QUESTION!!!

Do we really need 44100 samples per sec (frequencies between 0 and 22.1k?)

Even your laptop speakers can't pass these freqs, but you can crearly classify the stuff well, so maybe downsampling is a good idea?

QUESTION!!!

Why we split signal in 20 ms parts?

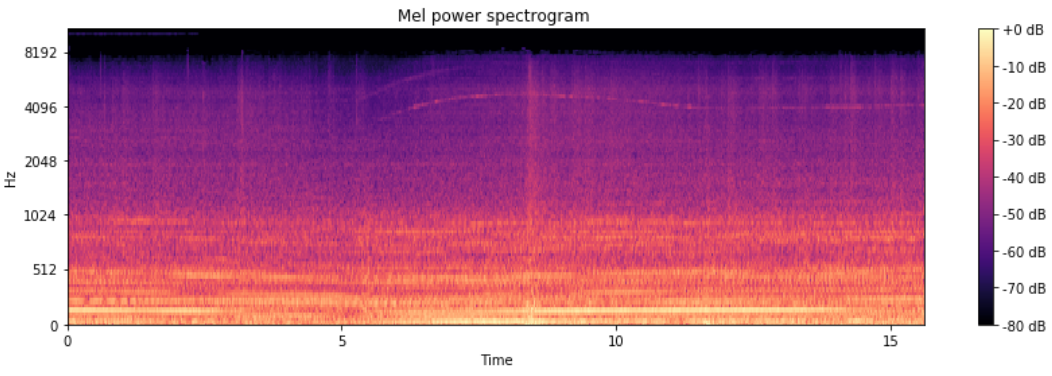
20 ms is a mean time of the shortest speech p**art, phonem. But wait, it's not a speech recognition competition, so maybe we should use longer windows? I think we should

MFCC

If you want to get to know some details about *MFCC* take a look at this great tutorial. [MFCC explained (http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/)](http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/) You can see, that it is well prepared to imitate human hearing properties.

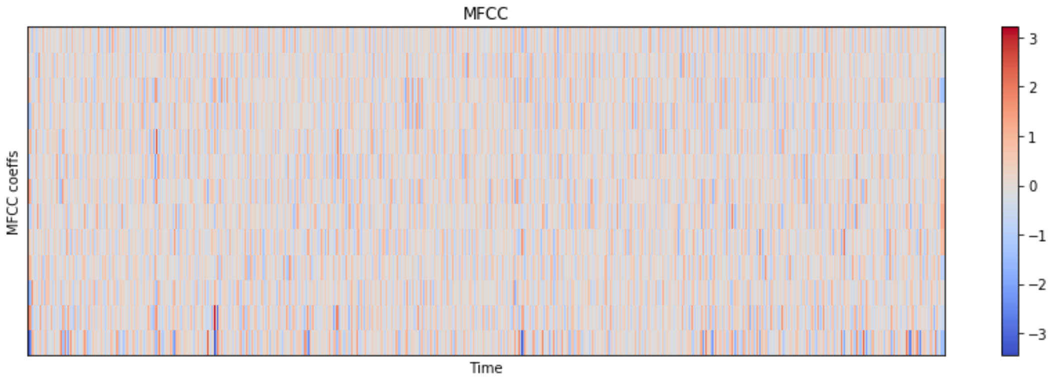
You can calculate *Mel power spectrogram* and *MFCC* using, for example, *librosa* python package.

In [11]:
```
# From this tutorial
# https://github.com/librosa/librosa/blob/master/examples/LibROSA%20demo.ipy
nb
S = librosa.feature.melspectrogram(samples1.astype(float), sr=sample_rate, n
_mels=128)

# Convert to log scale (dB). We'll use the peak power (max) as reference.
log_S = librosa.power_to_db(S, ref=np.max)

plt.figure(figsize=(12, 4))
librosa.display.specshow(log_S, sr=sample_rate, x_axis='time', y_axis='mel')
plt.title('Mel power spectrogram ')
plt.colorbar(format='%+02.0f dB')
plt.tight_layout()
```



In [12]:
```
mfcc = librosa.feature.mfcc(S=log_S, n_mfcc=13)

# Let's pad on the first and second deltas while we're at it
delta2_mfcc = librosa.feature.delta(mfcc, order=2)

plt.figure(figsize=(12, 4))
librosa.display.specshow(delta2_mfcc)
plt.ylabel('MFCC coeffs')
plt.xlabel('Time')
plt.title('MFCC')
plt.colorbar()
plt.tight_layout()
```

```
/opt/conda/lib/python3.6/site-packages/scipy/signal/_arraytools.py:45: Future
Warning:

Using a non-tuple sequence for multidimensional indexing is deprecated; use `
arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be interprete
d as an array index, `arr[np.array(seq)]`, which will result either in an err
or or a different result.
```



Explanation:

Human hearing is complex - we hear some frequencies more than other - and we can imitate it using the above techniques.

QUESTION!!!

Should we use MFCC?

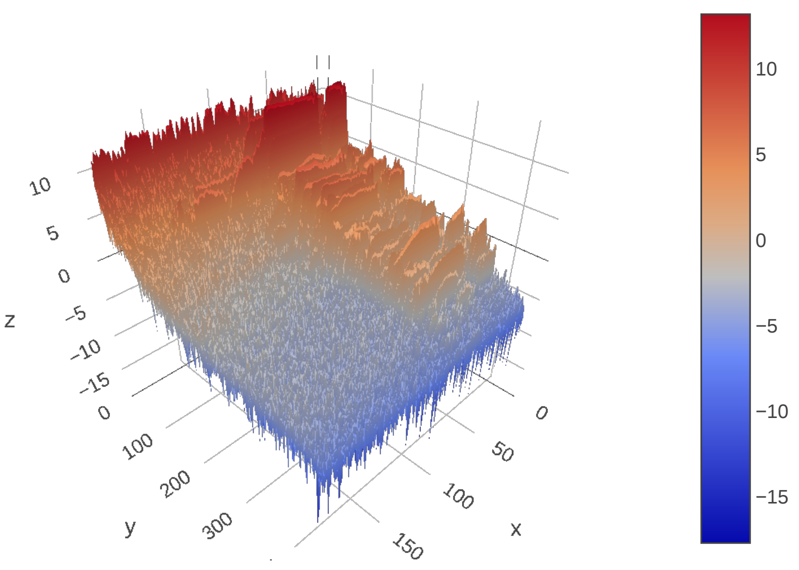
We evolved to here the environment and the speech, so I think we may not have the best detectors for some artificial signals, but I feel good at the classification with my natural filters, so I would USE MFCC.

And MFCC decorrelates the features! It's a nice property because the features are more distinct and tell more clear things.

Bonus - spectrogram in 3d

In [13]:
```
freqs, times, spectrogram = log_specgram(samples2, sample_rate)
data = [go.Surface(z=spectrogram.T)]
layout = go.Layout(
    title='Specgtrogram of "yes" in 3d',
#     scene = dict(
#     yaxis = dict(title='Frequencies', range=freqs),
#     xaxis = dict(title='Time', range=times),
#     zaxis = dict(title='Log amplitude'),
#     ),
)
fig = go.Figure(data=data, layout=layout)
py.iplot(fig)
```

Specgtrogram of "yes" in 3d



If you like my work please upvote.

Leave a feedback that will let me improve!

.

.

.