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 installed
# 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 list 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_submission.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

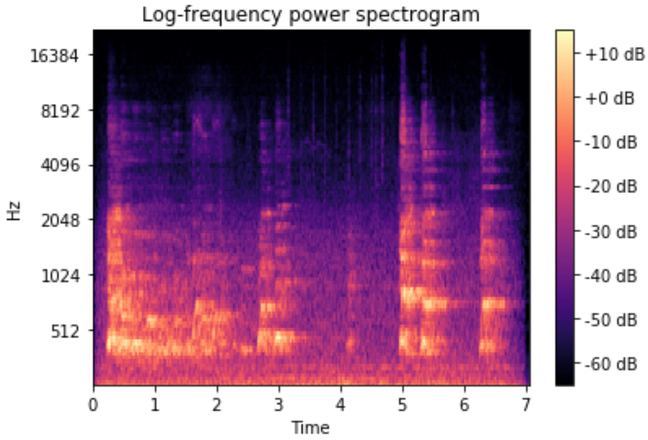
Almost copyed from my repository: https://github.com/daisukelab/ml-sound-classifier (https://github.com/daisukelab/ml-sound-classifier)

- 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
```

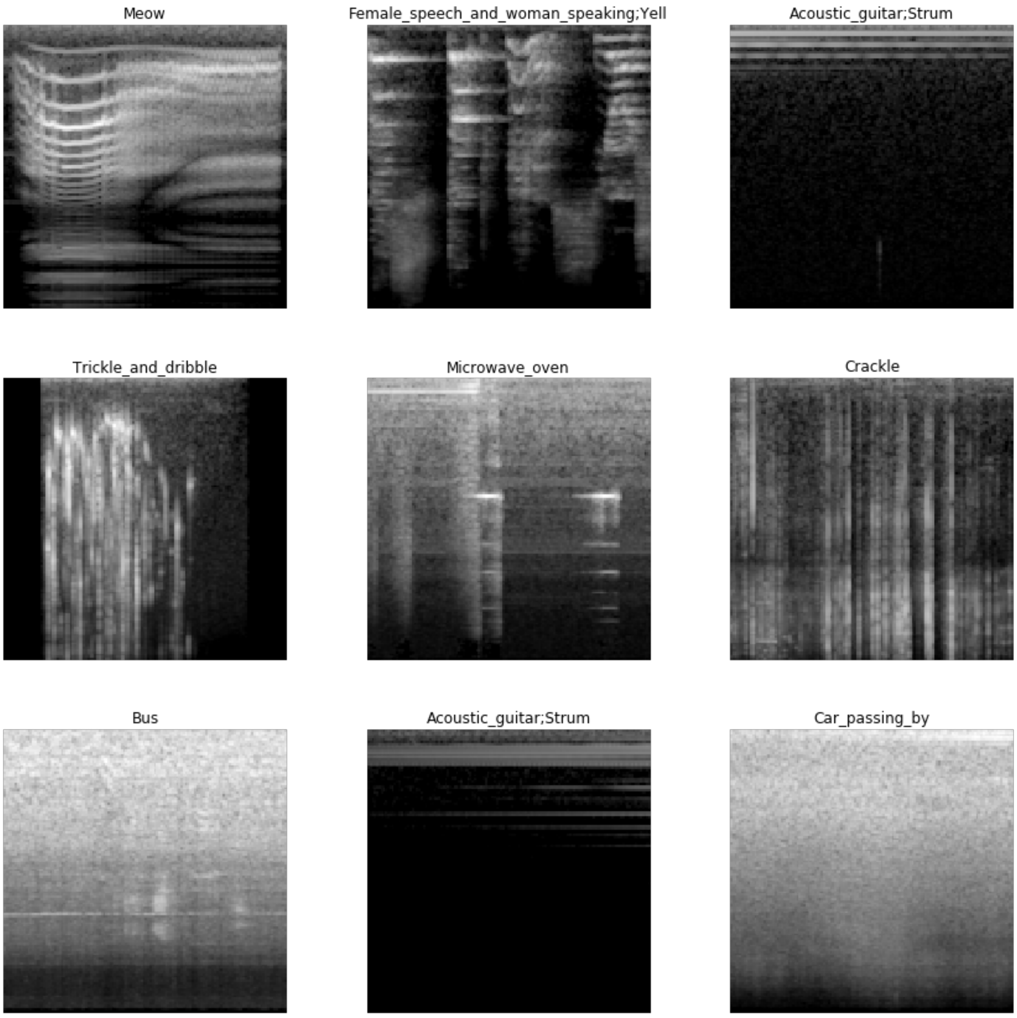
Follow multi-label classification

- 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)
- But `pretrained=False`
- With lwlrap as metric: 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):
```

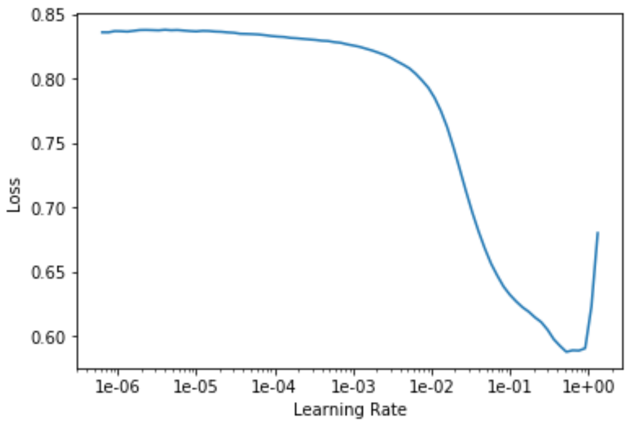
In [7]:
```
tfms = get_transforms(do_flip=True, max_rotate=0, max_lighting=0.1, max_zoo
m=0, max_warp=0.)
src = (ImageList.from_csv(WORK/'image', Path('../../')/CSV_TRN_CURATED, fold
er='trn_curated')
       .split_by_rand_pct(0.2)
       .label_from_df(label_delim=',')
)
data = (src.transform(tfms,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



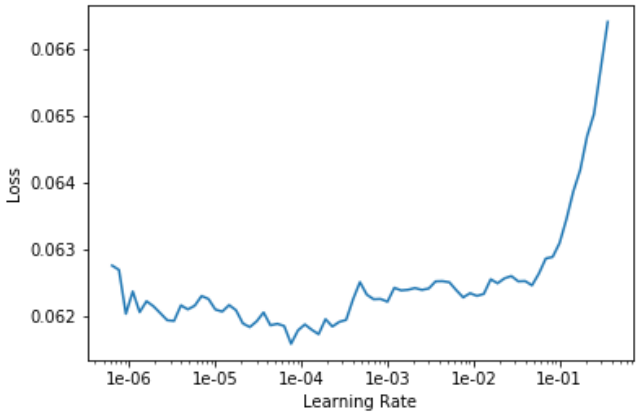
In [10]:
```
learn.fit_one_cycle(5, 1e-1)
learn.fit_one_cycle(10, 1e-2)
```

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.
```

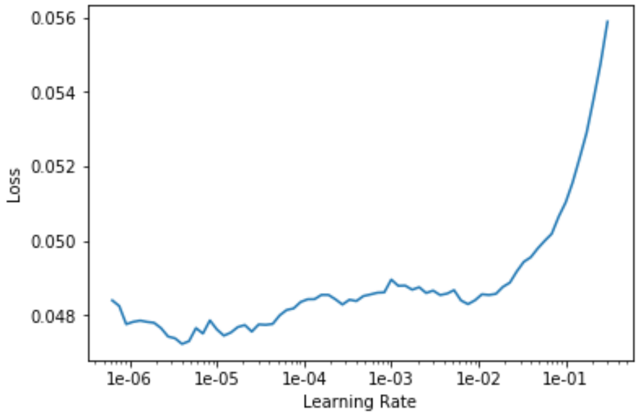


In [12]:
```
learn.fit_one_cycle(20, 3e-3)
```

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.
```

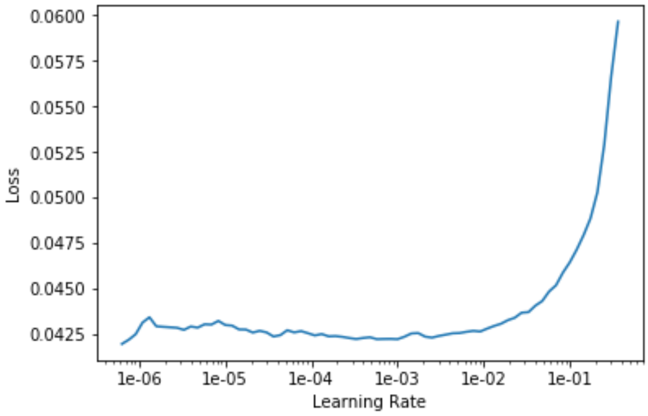


In [14]:
```
learn.fit_one_cycle(20, 1e-3)
```

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))
```

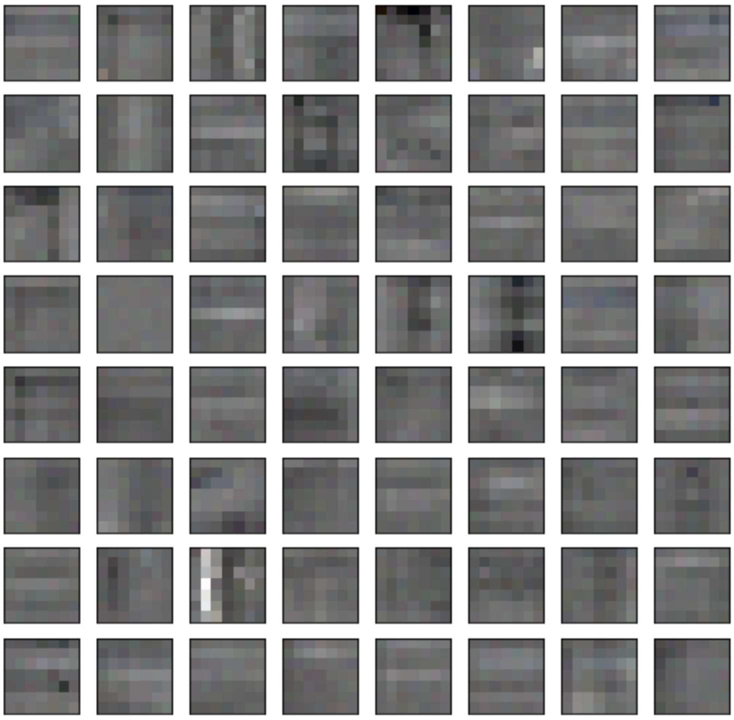
epoch	train_loss	valid_loss	lwlrap	time
0	0.042792	0.042987	0.591302	00:05
1	0.042829	0.043255	0.582411	00:05
2	0.042896	0.042537	0.597984	00:05
3	0.042663	0.042821	0.585062	00:05
4	0.042635	0.043676	0.573729	00:05
5	0.041898	0.042845	0.595862	00:05
6	0.042173	0.043182	0.591107	00:05
7	0.042239	0.043035	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()
```

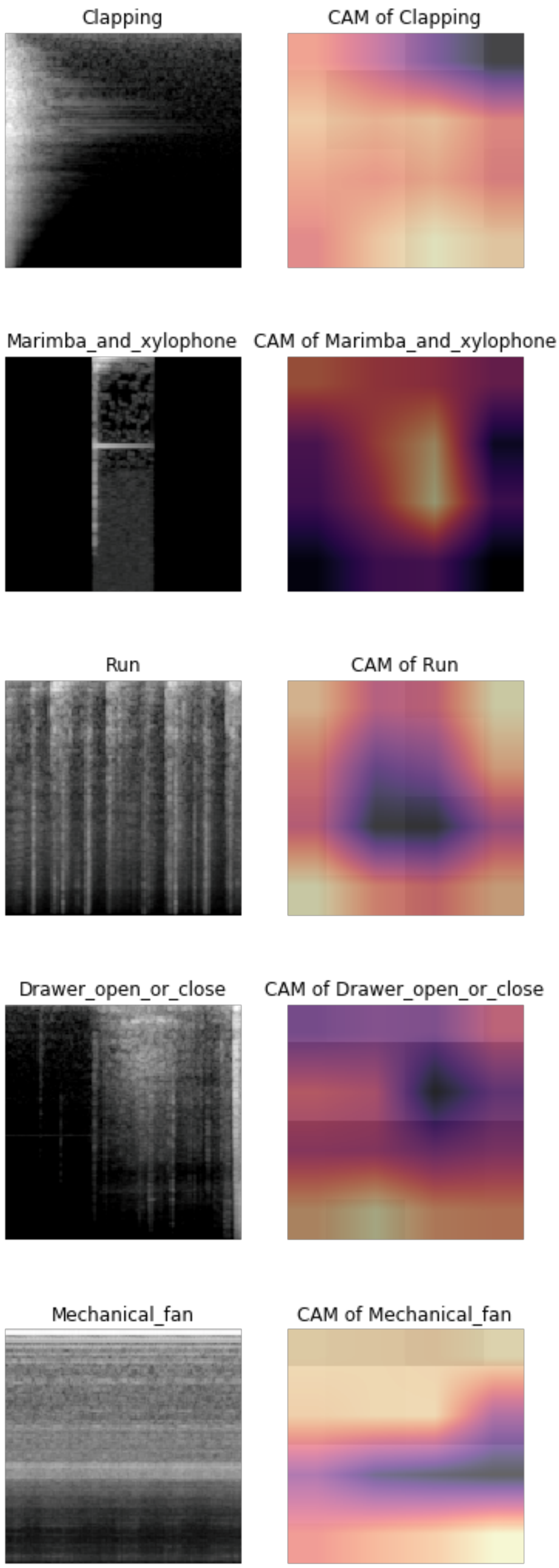
Out[21]:

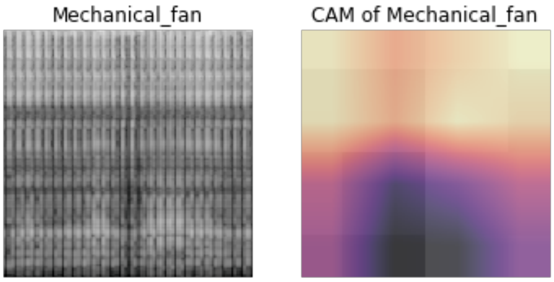
	fname	Accelerating_and_revving_and_vroom	Accordion	Acoustic_guitar	Applause	
0	000ccb97.wav	0.000013	9.970533e-08	3.473719e-08	1.873726e-07	1.3019
1	0012633b.wav	0.069694	3.735964e-05	9.331258e-05	1.868762e-04	2.7503
2	001ed5f1.wav	0.000024	6.408121e-05	4.881012e-07	5.531685e-05	8.6941
3	00294be0.wav	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

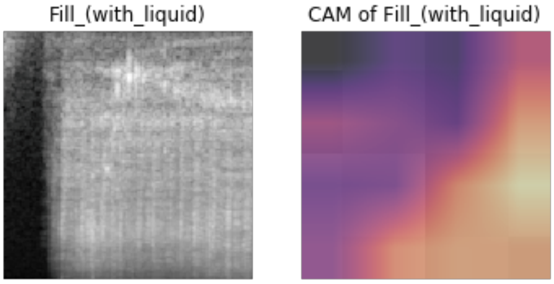
In [22]:
```
CUR_X_FILES, CUR_X = list(df.fname.values), X_train
learn = cnn_learner(data, models.resnet18, pretrained=False, metrics=[lwlra
p])
learn.load('fat2019_fastai_cnn2d_stage-2');
```

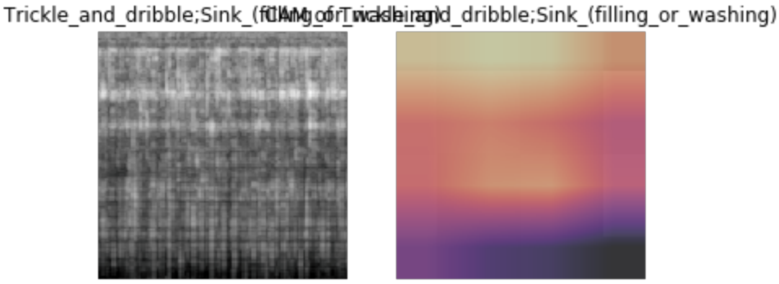
Visualize by CAM

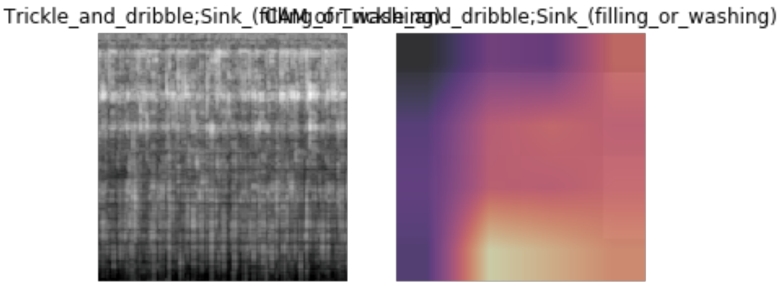
In [23]:
```
# Thanks to https://nbviewer.jupyter.org/github/fastai/course-v3/blob/master
/nbs/dl1/lesson6-pets-more.ipynb
from fastai.callbacks.hooks import *

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)
```

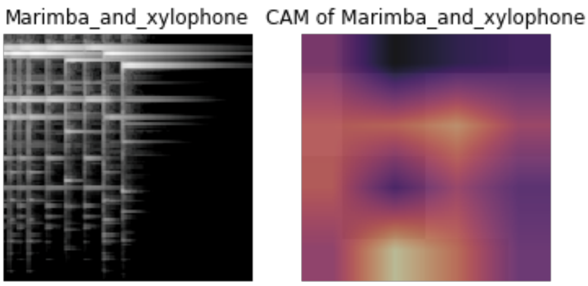

Clapping
CAM of Clapping
Marimba_and_xylophone
CAM of Marimba_and_xylophone
Run
CAM of Run
Drawer_open_or_close
CAM of Drawer_open_or_close
Mechanical_fan
CAM of Mechanical_fan

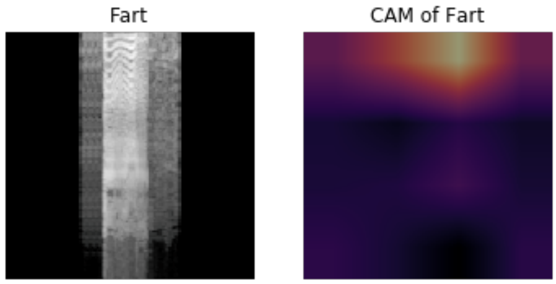

Mechanical_fan
CAM of Mechanical_fan


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]: