Machine Learning Engineer Nanodegree Capstone Project

Multi-Label Auto-Tagging of Audio Files Using fastai2 Audio

Mike Fuller, 19 April 2020

I. Definition

Project Overview

The sub-field of Machine Learning known as Machine Listening is a burgeoning area of research using signal processing for the automatic extraction of information from sound by a computational analysis of audio. There are many different areas of research within this field as demonstrated by the latest Detection and Classification of Acoustic Scenes and Events (DCASE) 2020 Challenge [1], a machine learning challenge dedicated to the research and development of new methods and algorithms for audio. These include:

- Acoustic Scene Classification
- Sound Event Detection and Localization
- Sound Event Detection and Separation in Domestic Environments
- Urban Sound tagging
- Automated Audio Captioning

As an acoustic engineer, I am extremely intrigued by this new field. Recent developments in machine learning algorithms have allowed significant progress to be made within this area, with the potential applications of the technology being wide and varied and meaning the tools could prove to be extremely useful for the acoustic practitioner amongst many other uses.

The in-development user-contributed fast.ai2 Audio library [2] inspired me to undertake the development of a deep learning audio-tagging system for this Udacity Capstone project, described herein.

Problem Statement

The Freesound Audio Tagging 2019 Kaggle Competition provides the basis for my research project [3].

The challenge is to develop a system for the automatic classification of multi-labelled audio files within 80 categories, that could potentially be used for automatic audio/video file tagging with noisy or untagged audio data. This has historically been investigated in a variety of ways:

- Conversion of audio to mel-spectrogram images fed into CNNs
- End-to-End Deep learning
- Custom architectures involving auto-encoders

- Features representation transfer learning with custom architectures and Google's Audioset

In addition, the classification of weakly labelled data from large-scale crowdsourced datasets provides a further problem for investigation[4]. The problem is clearly quantifiable in that a number of accuracy metrics could be used to quantify the accuracy of the model's predictions, described below.

The competition dataset comprises audio clips from the following existing datasets:

- "Curated Train Set" - Freesound Dataset (FSD): a smaller dataset collected at the MTG-UPF based on Freesound content organized with the AudioSet Ontology and manually labelled by humans. 4964 files.
- "Noisy Train Set" - The soundtracks of a pool of Flickr videos taken from the Yahoo Flickr Creative Commons 100M dataset (YFCC) which are automatically labelled using metadata from the original Flickr clips. These items therefore have significantly more label noise than the Freesound Dataset items. 19815 files.

The data comprises 80 categories labelled according to Google's Audioset Ontology[3] with ground truth labels provided at the clip level. The clips range in duration between 0.3 to 30s in uncompressed PCM 16 bit, 44.1 kHz mono audio files.

Solution Statement

With the above competition requirements in mind, the proposed solution was followed and was undertaken initially within a Pytorch AWS SageMaker notebook instance using Jupyter Notebooks, and further, using a Google Cloud Platform AI Notebook using fastai2 and fastai2 audio libraries , due to the extra credits required for the long training times on a GPU instance:

1. The data will be downloaded from Kaggle into the chosen platform AWS SageMaker / GCP AI Notebook instance
2. Exploratory Data Analysis - The dataset will be downloaded such that the file metadata can be extracted, in order to confirm: sample rates, bit-rates, durations, channels (mono/stereo) for each file in order to direct initial signal processing stage and approach towards the dataset splitting. In addition, the statistics of the file labels will be analysed.
3. Model Development:
    - The fastai2 and fastai2 audio libraries will be installed
    - The fastai2 audio library will be used for the data processing, in order to convert the audio files into tensor representations of mel-spectrograms on-the-fly, rather than in a separate pre-processing stage. This is a significant benefit of the library in terms of allowing quick experimentation and iteration within the model development over other methods such as converting all audio files to mel-spectrogram images separately.
    - In-line with the competition rubric, a non-pretrained convolutional neural network (CNN) using the fastai2 library for PyTorch will be developed using state-of-the-art methods and additions.
    - The model will be trained on the "Noisy Train Set" in a 5-Folds Cross Validation manner, using Sci-Kit Learn's K-Fold model selection module[5], for a shorter period due to the large amount of data.
    - The results of these 5 models will be used to train on the "Curated Train Set" in the same 5-Folds Cross Validation manner as the noisy train set, but for a longer period.
    - Test-Time-Augmentation (TTA) will be used to gain averaged predictions from all 5 final models on the test set. The predictions will be submitted as a Late-Submission for the analysis of the results.

- This will be repeated, with tweaks to the model augmentations in order to try to improve the results iteratively.

Metrics

Due to the advancement of multi-label audio classifiers in recent years, a simple multi-label accuracy metric was not used within the Kaggle competition, as performances of the systems can easily exceed 95% within a few epochs of training.

As such, the competition used label-weighted label-ranking average precision (a.k.a lwl-rap) as the evaluation metric. The basis for the metric, the label-ranking average precision algorithm, is described in detail within the Sci-Kit Learn implementation [6]. The additional adaptations of the metric are to provide the average precision of predicting a ranked list of relevant labels per audio file, which is a significantly more complex problem to solve than a standard multi-label accuracy metric. The overall score is the average over all the labels in the test set, with each label having equal weight (rather than equal weight per test item), as indicated by the "label-weighted" prefix. This is defined as follows [7]:

$$lwlrap = \frac{1}{\sum_s |C(s)|} \sum_a \sum_{\epsilon\epsilon\ C(s)} Prec(s, c)$$

where $Prec(s, c)$ is the label-ranking precision for the list of labels up to class c and the set of ground-truth classes for sample s is $C(s)$. $|C(s)|$ is the number of true class labels for sample s.

The Kaggle competition provides a Google Colab example implementation [8].

II. Analysis

Data Exploration

The datasets were downloaded from Kaggle using the Kaggle API and analysed within a Jupyter Notebook (see: eda.ipynb within the nbs_final folder).

The first stage of the process was to understand the dataset more fully. Fortunately, due to being a Kaggle Competition dataset it was well documented and clean in terms of organization.

Downloading the dataset was undertaken using guidance given within the Kaggle Forums [9] directly into the SageMaker/GCP Instance storage for easy access.

The files were then unzipped for the EDA. For further details, please see the notebook directly.

Pandas and Pandas Profiling

In order to undertake the analysis of the data, the numerical data analysis packages Pandas and Pandas Profiling were used.

Pandas Profiling [10] is an extremely useful add-on package to Pandas, which creates HTML profile reports directly from Pandas DataFrames quickly and easily. From the provided .csv files file category labels were analysed and, in addition, the audio file meta-data was extracted (i.e. sample rates, bit-rates, durations, number of channels).

	fname	labels	duration,s	sample rate,Hz	channels	bits	encoding
0	0006ae4e.wav	Bark	7	44100	1	16	sox_encoding_t.SOX_ENCODING_SIGN2
1	0019ef41.wav	Raindrop	2	44100	1	16	sox_encoding_t.SOX_ENCODING_SIGN2
2	001ec0ad.wav	Finger_snapping	2	44100	1	16	sox_encoding_t.SOX_ENCODING_SIGN2
3	0026c7cb.wav	Run	26	44100	1	16	sox_encoding_t.SOX_ENCODING_SIGN2
4	0026f116.wav	Finger_snapping	1	44100	1	16	sox_encoding_t.SOX_ENCODING_SIGN2

Fig 1. An example Pandas DataFrame of extracted audio file info

Using these two packages the following was found:

Curated Train Data

For the Curated Train dataset, it was found that the bit-rate was a constant 16bits, the channels a constant 1 (mono), constant sample rate of 44100kHz and that there were 213 different tagging combinations of the 80 audio labels over the total file count (4964 files):

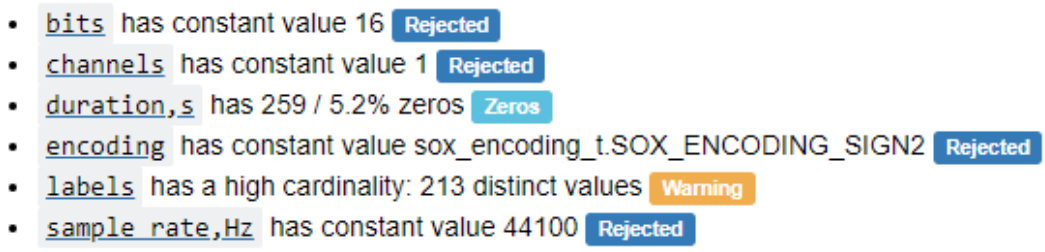

Fig 2. Pandas Profiling for the Curated Train Data set

In terms of the file durations, the average file length was 7.63 seconds and the files ranged between just over 0 and 30 seconds long, with the lengths predominantly in the 0-5 seconds length range. This will affect the mel-spectrogram processing of the data, i.e. we will need to ensure a sufficient amount of both the longer and smaller audio files are taken, in order for the feature learning of the CNN to be accurate.

Fig 3. Pandas Profiling information for the audio file durations

Noisy Train Data

As with the Curated dataset, with the Noisy Train dataset it was found that the bit-rate was a constant 16bits, the channels a constant 1 (mono), constant sample rate of 44100kHz. However, in this dataset there were 1168 different tagging combinations of the 80 audio labels over the total file count (19815 files):

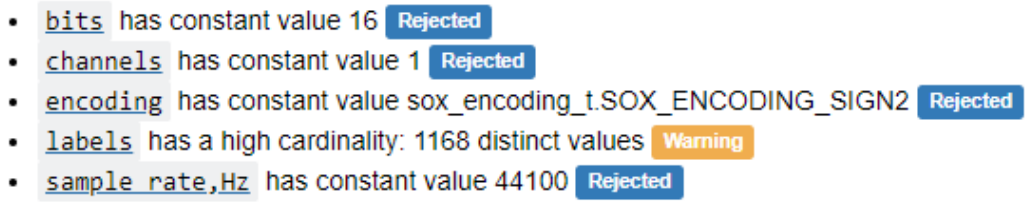

Fig 4. Pandas Profiling for the Noisy Train dataset

The Noisy Train dataset average file length was significantly longer on average than the Curated set at 14.6s long, however, the files ranged between 1 and 16 seconds long. There is therefore a significant difference in terms of length between the two datasets.

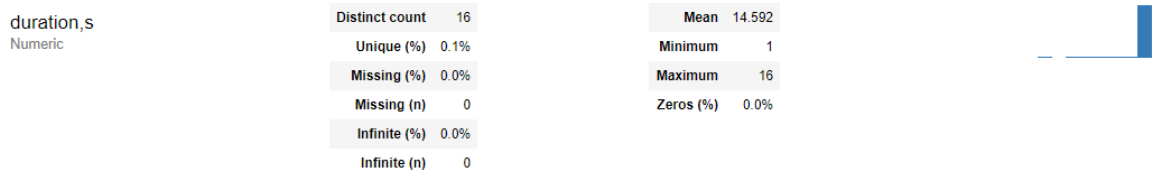

Fig 5. Pandas Profiling information for the audio file durations

In addition, as the name implies, the Noisy Train set files have a significantly higher noise floor than the Curated Train set due to the provenance of the files.

Data Visualisation

The following figure clearly illustrates the differences between the difference in durations of audio files between the two datasets:

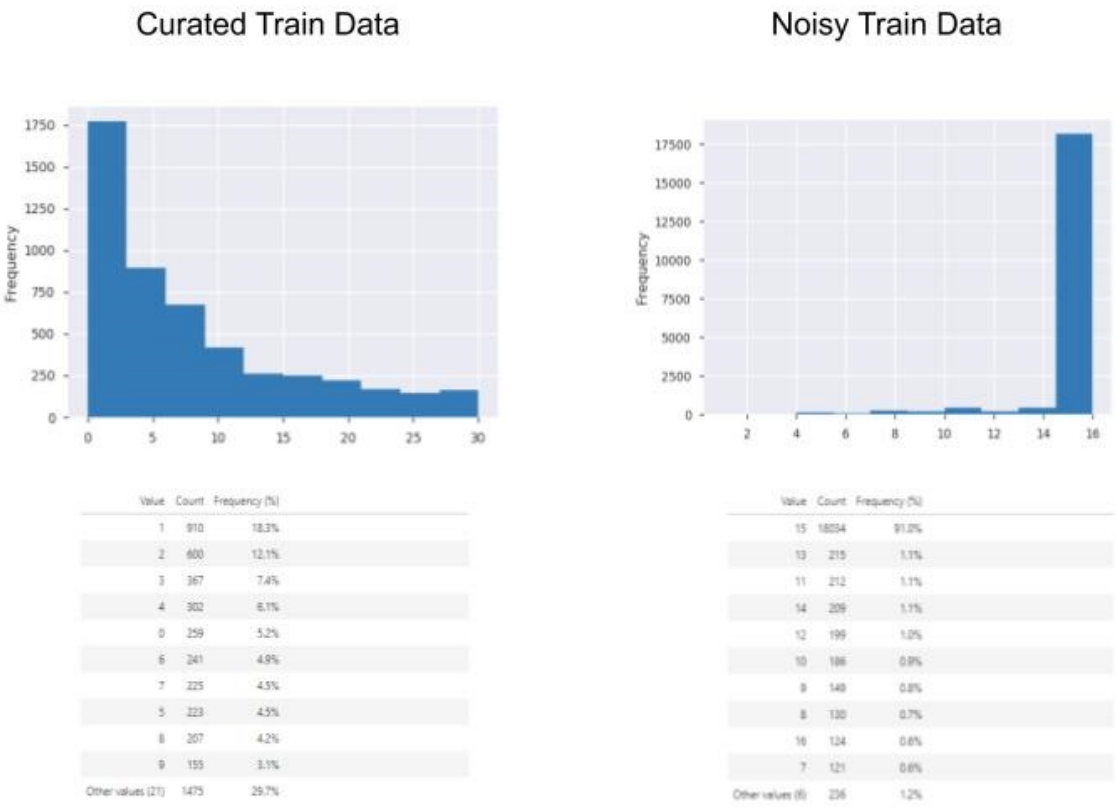

Fig 6. Train vs Noisy dataset durations (x-axis = seconds)

Therefore, in the development of the model the following factors will need to be considered:

- Noise floor differences between the curated and noisy train set will affect how the signals are clipped to shorter lengths to feed into the CNN.
- The average lengths also have a high range of values both over the the individual datasets and between the curated and noisy set, we will need to ensure the main recorded features corresponding to the file labels of each recording are kept within any clipped sections to produce the mel-spectrograms.

Algorithms and Techniques

Mel-Spectrograms

This signal processing stage will involve trimming (to ensure uniform duration) in order be converted to uniform length log-mel-spectrogram representations of the audio. A log-mel-spectrogram is a spectrogram representation of the audio i.e. a frequency-domain representation based on the Fourier Transform, with x-axis = time, y axis = frequency and colour depth/pixel value = relative sound intensity, which has been has been converted to the Mel scale on the y-axis by a non-linear transform in order to be more representative of the highly non-linear magnitude and frequency sensitivities of the human ear [11]. The chosen settings will be discussed and shown further in the Data Preprocessing section.

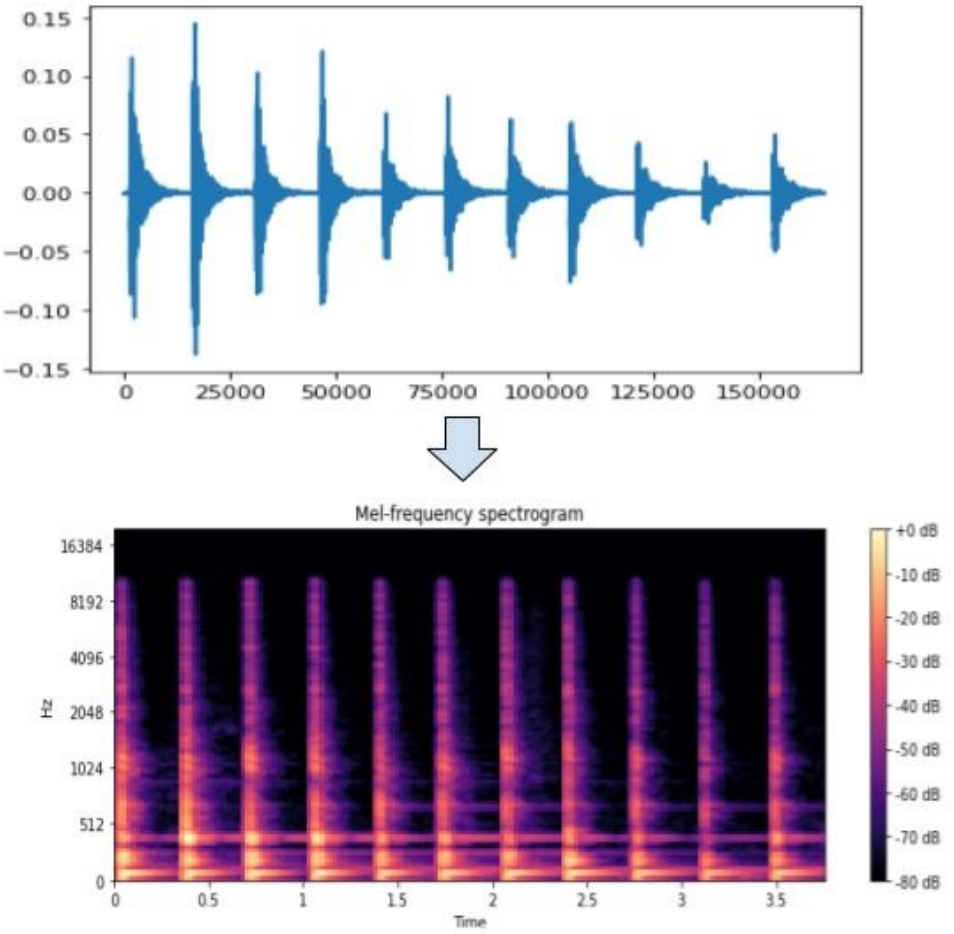

Fig 7. Conversion from Waveform to Mel-spectrogram representation

Convolutional Neural Network (CNN)

The length uniformity of the audio clips in is important, as it allows 2D tensors of the mel-spectrograms to be fed in batches into a CNN. The model variety used was as follows, based on the state of the art findings of the fastai community [12] and other research described below. The model and architecture used the following settings:

- Architecture: fastai2's XResNet50 based on the Bag of Tricks [13] research paper which includes tweaks to the optimisation methods for higher performance. ResNets use skip connections in order to allow propagation of information more deeply architecture, giving significant speed improvements for deeper networks and allowing the gradient descent to

backpropagate through the network which aids in increasing training accuracy. This has further been augmented in the Bag of Tricks paper, whereby the residual block convolutional layers have been re-arranged such that further efficiency gains are made.

- Activation Function: Mish [14] which has been shown to provide performance improvements over the standard ReLU activation function due to its smoothing of the activations rather than the cut-off of the ReLU function for values below 0.
- Optimizer Function: Ranger which is a combination of the RAdam [15] and Lookahead [16] optimizer functions. These functions work as a searching pair, whereby one learner goes ahead of the other to explore the function topography, such that traps involving local minima can be avoided.
- Layer tweaks: Self-Attention Layers [17]
- Replacing Max Pooling Layers with "MaxBlurPool" layers for better generalization
- Flat-Cosine decay learning rate scheduling

K-Folds Validation

Sci-Kit Learn's KFolds Validation function was used to split the datasets into 5 folds, to allow all of the available data to be used in the training and to further allow the 5 created models to give ensembled predictions on the Test set, which provides a significant performance improvement over a single model.

MixUP

MixUp, whereby two spectrograms are combined to form a third, was also used during the longer Curated Training Set procedure. Detailed further below.

Test-Time Augmentation (TTA)

In addition to the methods outlined above, Test-Time augmentations were applied to the test set, such that the data transformations were used as part of the testing procedure in order to give a further performance boost.

Benchmark

The Baseline performance for the Kaggle Competition was set at 0.53792 which provided a minimum target. The winner [18] of the competition acheived 0.75980, which provided the upper target. The details of the winning model and training method can be found on the linked GitHub page, but for brevity, the basic details of the system from the GitHub repo, were as follows:

> - Log-scaled mel-spectrograms
> - CNN model with attention, skip connections and auxiliary classifiers
> - SpecAugment, Mixup augmentations
> - Hand relabeling of the curated dataset samples with a low score
> - Ensembling with an MLP second-level model and a geometric mean blending

III. Methodology

Data Preprocessing

Fastai2 Audio

The fastai2 audio package was used to convert the audio files into mel-spectrograms 2D tensors on-the-fly, as a form of efficient data processing rather than pre-processing and saving to a different dataset. This was done using the following process:

1. Create Pandas DataFrames for the files, suing the `train_curated.csv` and `train_noisy.csv` files provided by the competition, removing corrupted or empty files as given in the competition guidance:

```
def create_train_curated_df(file, remove_files=[]):
    df_curated = pd.read_csv(file)
    df_curated['fname'] = '../data/train_curated/' + df_curated['fname']
    df_curated.set_index('fname', inplace=True)
    df_curated.loc[remove_files]
    df_curated.drop(index=remove_files, inplace=True)
    df_curated.reset_index(inplace=True)
    return df_curated

def create_train_noisy_df(file):
    df_noisy = pd.read_csv(file)
    df_noisy['fname'] = '../data/train_noisy/' + df_noisy['fname']
    return df_noisy


# Create Curated training set df
# Remove corrupt and empty files as per Kaggle guidance

remove_files = ['f76181c4.wav', '77b925c2.wav', '6a1f682a.wav', 'c7db12aa.wav',
'7752cc8a.wav', '1d44b0bd.wav']
remove_files = ['../data/train_curated/' + i for i in remove_files]
df_curated = create_train_curated_df('../data/train_curated.csv',
remove_files=remove_files)
df_curated.head()
```

The DataFrames were then used to supply the fastai DataBlock API with the filenames, which could then be processed using the fastai2 audio `item_transformations` which are applied to each file before training. After significant testing iterations the audio transformations settings were chosen as follows:

```
DBMelSpec = SpectrogramTransformer(mel=True, to_db=True) # convert to Mel-spectrograms

clip_length = 2 # clip subsection length in seconds
sr = 44100 # sample rate
f_min = 20 # mel-spectrogram minimum frequency
f_max = 20000 # mel-spectrogram minimum frequency
n_mels = 128 # mel-frequency bins, dictates the y-axis pixel size
hop_length = math.ceil((clip_length*sr)/n_mels)# determines width of image. for square to match n_mels, set math.ceil((clip_length*sr)/n_mels)
nfft = n_mels * 20 # = 2560 for higher resolution in y-axis
win_length = 1024 # sample windowing
top_db = 60 # highest noise level in relative db
```

- The `top_db` parameter setting of 60dB was important, as the noisy train set had high background noise which with a higher setting obscured some of the frequency features.

In addition to the mel-spectrogram settings above, the following additional item transformations were undertaken:

- `RemoveSilence` - Splits the original signal at points of silence more than 2 * pad_ms
- `CropSignal` - Crops a signal by clip_length and adds zero-padding by default if signal is less than clip_length
- `aud2spec` - The mel-spectrogram settings from above
- `MaskTime` - Uses Google's SpecAugment [19] time masking procedure to zero-out time domain information as a form of data augmentation
- `MaskFreq` - Uses Google's SpecAugment [19] frequency masking procedure to zero-out frequency domain information as a form of data augmentation

```
item_tfms = [RemoveSilence(threshold=20),
             CropSignal(clip_length*1000),
             aud2spec,
             MaskTime(num_masks=1, size=8), MaskFreq(num_masks=1, size=8)]
```

Batch Transforms

In addition to the item transforms above, Batch Transforms were used as part of the DataBlock API, which are transformations applied per batch during training:

- `Normalize()` - normalizes the data taking a single batch's statistics
- `RatioResize(256)` - during training (other than the first 10 epochs of the noisy data for speed), the mel-spectrogram tensors were resized from 128x128px to 256x256px through bilinear interpolation as this has been shown to give gains in performance over simply creating 256x256px from the outset.
- `Brightness and Contrast` augmentations were also applied in the training cycles to improve performance

```
batch_tfms = [Normalize(),
              RatioResize(256),
              Brightness(max_lighting=0.2, p=0.75),
              Contrast(max_lighting=0.2, p=0.75)]
```

The above augmentations (prior to batch transformations), produced the following mel-spectrograms as 2D tensors, plotted via matplotlib:

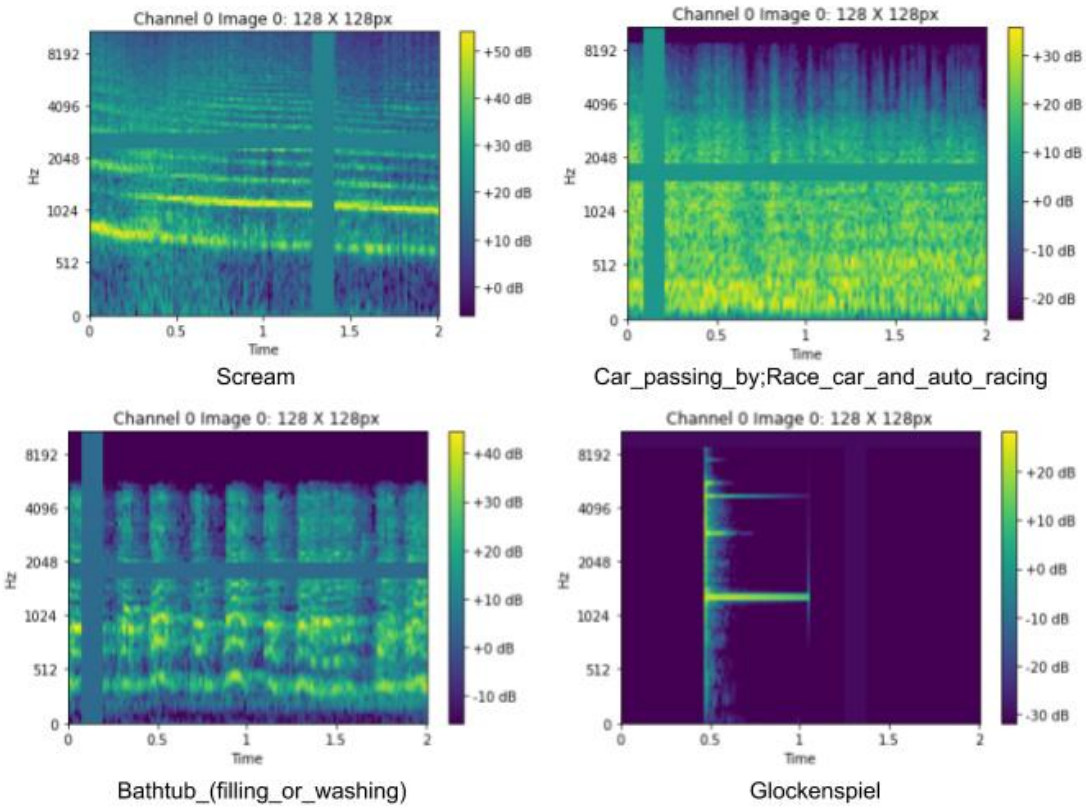

Fig 8. Augmented Mel-spectrograms

Implementation

The data augmentations stated above were used to significantly improve the performance of the classifier during the following K-Fold training cycles.

The implemented training method was chosen based on the Competitions 6th place winner's technique [19], however, only the first two stages were implemented as follows due to the cost requirements using GCP:

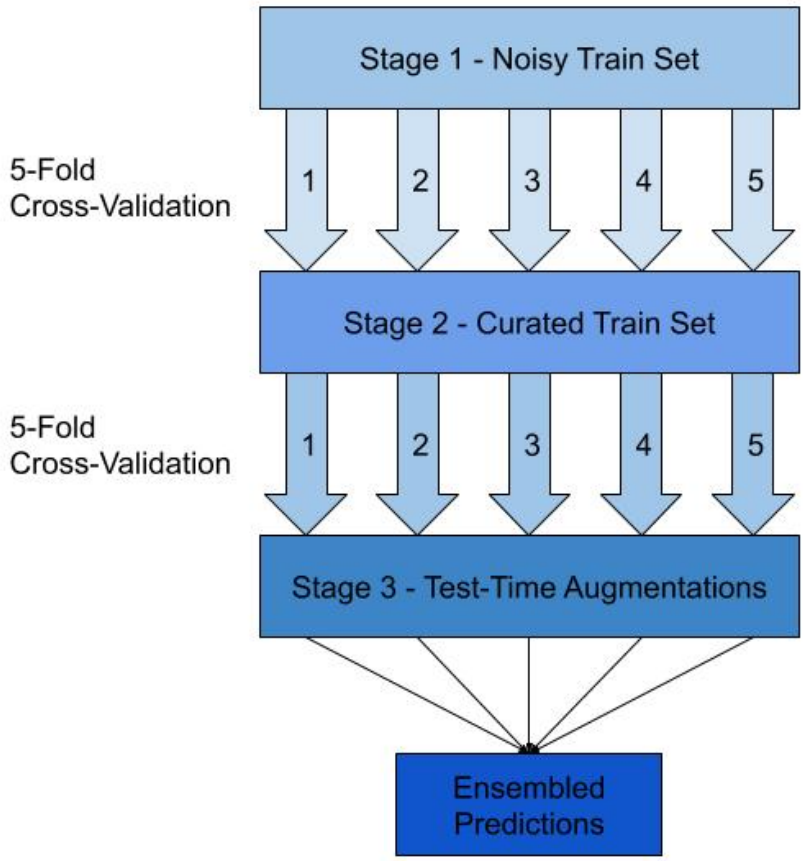

Fig 9. Train-Test-Prediction Stages

Stage 1 - Noisy Training Set

As can be seen below, the following 5-Fold training cycle was used on the noisy set. The indices of the DataFrame were shuffled to ensure the data splits were chosen at random, but without overlap using SKLearns k-Folds module. The cycle began with 10 epochs of training at a higher learning rate and then 10 epochs of training at a lower learning rate (set after using fastai's learning rate finder during the testing stage) used to fine-tune the model's weights further. Please see the associated Jupyter Notebook for the training output.

The models were then saved for further training on the curated training set.

Note: No MixUp augmentations were used on the Noisy Training set.

```
from sklearn.model_selection import KFold

# Declare Number Folds
n_splits = 5

kf = KFold(n_splits=n_splits, random_state=42, shuffle=True) # random_state for
repeatable results, shuffle indices
```

```
df = df_noisy # to use random subset, use  df = df_.sample(frac=0.5,
replace=False, random_state=1) # take random subset of the noisy dataframe for
faster training (otherwise need 6.5 hours for all folds with complete dataset)

for fold, (train_idx, valid_idx) in enumerate(kf.split(df)):
    print(f'\nNoisy Train Set - Fold {fold+1}/{n_splits}')

    def get_x(r): return r['fname']
    def get_y(r): return r['labels'].split(',') # split labels on ','

    def get_dls(train_cycle):
        if train_cycle == 1:
            batch_tfms = [Normalize(),
                          Brightness(max_lighting=0.2, p=0.75),
                          Contrast(max_lighting=0.2, p=0.75)]

        elif train_cycle == 2:
            batch_tfms = [Normalize(),
                          RatioResize(256), # progressive resize to 256x256px
                          Brightness(max_lighting=0.2, p=0.75),
                          Contrast(max_lighting=0.2, p=0.75)]

        dblock = DataBlock(blocks=(AudioBlock, MultiCategoryBlock),
                           splitter=IndexSplitter(valid_idx), # split using df
index
                           get_x=get_x,
                           get_y=get_y,
                           item_tfms = item_tfms,
                           batch_tfms = batch_tfms
                          )
        return dblock.dataloaders(df, bs=64)

    dls = get_dls(train_cycle=1)

    dls.show_batch(max_n=6)

    model = xresnet50(pretrained=False, act_cls=Mish, sa=True, c_in=1, n_out=80)
#create custom xresnet: 1 input channel,  80 output nodes, self-attention, Mish
activation function
    model = convert_MP_to_blurMP(model, nn.MaxPool2d) # convert MaxPool2D layers
to MaxBlurPool
    learn = Learner(dls, model=model, loss_func=BCEWithLogitsLossFlat(),
opt_func = ranger, metrics=[lwlrap]) # pass custom model to Learner, no mixup
for noisy set as fewer epochs

    learn.fit_flat_cos(10, lr=3e-3)

    print('Batch transforming images to 256x256px and training further.')

    dls = get_dls(train_cycle=2)
    learn.dls = dls
    learn.fit_flat_cos(10, lr=3e-3/3)

    print('Saving Learner...')
    learn.save(f'stage-1_noisy_fold-{fold+1}_sota2')
```

Stage 2 - Curated Train Set

After all 5 models had been trained on the Noisy set, the models were then trained on different 5-folds of the Curated Set. This essentially gave 5 distinct models, all trained on different data for later ensembling.

Note: MixUp data augmentations were applied to the Curated Train set, shown as training callback below. This is whereby two spectrogram tenors are combined into a single 2D tensor with a certain percentage blend (50% in this case), allowing the network to learn double the amount of features and labels per batch. This also provides a form of regularization for the model which improves generalization on the validation/test sets.

```
## K-Folds training loop

df = df_curated

for fold, (train_idx, valid_idx) in enumerate(kf.split(df)):
    print(f'\nCurated Train Set - Fold {fold+1}/{n_splits}')

    def get_x(r): return r['fname']
    def get_y(r): return r['labels'].split(',') # split labels on ','

    dblock = DataBlock(blocks=(AudioBlock, MultiCategoryBlock),
                       splitter=IndexSplitter(valid_idx), # split using df index
                       get_x=get_x,
                       get_y=get_y,
                       item_tfms = item_tfms,
                       batch_tfms = batch_tfms # including RatioResize(256)
                      )

    dls = dblock.dataloaders(df, bs=64)

    dls.show_batch(max_n=3)

    print(f'\nLoading Stage 1 model - fold {fold+1}.')

    model = xresnet50(pretrained=False, act_cls=Mish, sa=True, c_in=1, n_out=80)
#create custom xresnet: 1 input channel,  80 output nodes, self-attention, Mish activation function
    model = convert_MP_to_blurMP(model, nn.MaxPool2d) # convert MaxPool2D layers to MaxBlurPool
    learn = Learner(dls, model=model, loss_func=BCEWithLogitsLossFlat(),
 opt_func=ranger, metrics=[lwlrap]) # pass custom model to Learner, no mixup for noisy set as fewer epochs
    learn.load(f'stage-1_noisy_fold-{fold+1}_sota2')

    learn.dls = dls
    learn.add_cb(MixUp()) # add mixup callback

    print('\nTraining on Curated Set:')
    learn.fit_flat_cos(50, 3e-4)

    print('Saving model...')
    learn.save(f'stage-2_curated_fold-{fold+1}_sota2')
```

Testing

At the testing stage, Test-Time-Augmentations and ensembling the predictions of all 5 different Stage-2 models were used to improve the final predictions.

```
# grab test filenames from submission csv
df_fnames = pd.read_csv('../data/sample_submission.csv')
fnames = df_fnames.fname
df_fnames = '../data/test/' + df_fnames.fname
print(df_fnames[:5])

# get predictions
for fold in range(n_splits):
    stage = 2
    print(f'Getting predictions from stage {stage} fold {fold+1} model.')

    learn = learn.load(f'stage-2_curated_fold-{fold+1}_sota2')

    dl = learn.dls.test_dl(df_fnames)

    # predict using tta
    preds, targs = learn.tta(dl=dl)
    preds = preds.cpu().numpy()

    if fold == 0:
        predictions = preds
    else:
        predictions += preds


# Average predictions
predictions /= n_splits

# Create Submission DataFrame
df_sub = pd.DataFrame(predictions)
df_sub.columns = learn.dls.vocab
df_sub.insert(0, "fname", 0)
df_sub.fname = fnames
df_sub.head()
```

The produced .csv file was then submitted to Kaggle.

Refinement

Initial Model

The initial CNN architecture used was a pre-trained (on ImageNet) xresnet50 model for speed of iteration. This was trained on a single fold smaller subset of the Noisy data (ranging from 20-80% using the DataBlock API - `RandomSubsetSplitter()` function) used for faster iteration on the noisy subset, while all of the Curated data was used. The data augmentation settings were slightly different however, as non-square mel-spectrograms were used to see if larger spectrograms could give improved scores, which was the case, however, this was at the expense of training time.

This highest score achieved by any initial model, was an lwl-rap of 0.61013 on the test-set:

Fig 10. Initial Best Score

This score, while not bad for a small amount of testing and still beating the competition baseline, was far from achieving near state-of-the-art performance.

Test-Time-Augmentation was shown to provide a benefit of >3% improvement during further testing rounds using a single training fold on the noisy and curated datasets, shown as the top score in the following image:

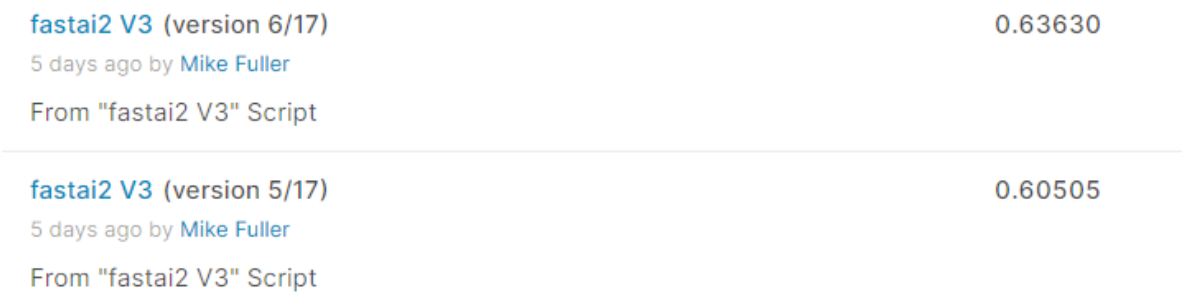

Fig 11. Improvement using TTA

After reading further the writeups of the competition winners and high scorers [18] [20] [21], it was decided that a K-Folds validation approach was required in order to substantially improve the performance.

In addition, due to the large size of the spectrograms in the initial testing phase that would cause extremely slow training over so many folds (a 5x increase in epochs), these were replaced with smaller 128x128px (using 128 mel-bins and the settings shown in the above Section III: Data Preprocessing section). It was decided to first try training for a single fold on the Noisy set (90%/10% train/test split) and then split this model into 5 separate models for the further training on the Curated Set.

Whatsmore, the pretrained xresnet50 model was replaced by the state-of-the-art xresnet50 model described above in Section II: Algorithms and Techniques. This was in-line with the allowance of only non-pretrained models in the competition and was also shown to provide small improvements over the pretrained xresnet50 over long enough training cycles, such that the non-trained units could effectively learn, as shown below:

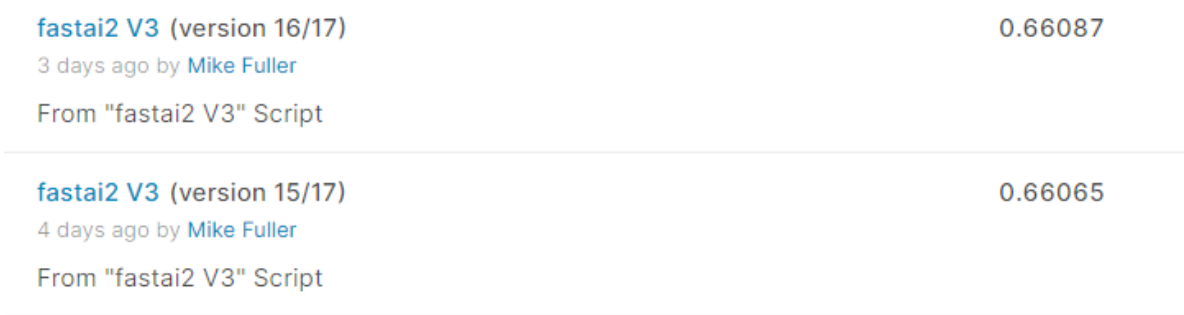

Fig 12. Improvement using SOTA model

Finally, a full 5-Fold Cross-Validation training was choundertaken for both the Noisy and Curated set as detailed in Figure 9 in Section III: Implementation above, with some tweaks to the spectrograms settings, i.e. using `top_dB` of 60 to ensure only the most prominent Noisy Set features were captured by the mel-spectrograms. This this approach achieved the final score of 0.69788, a marked improvement that would have gained a bronze-medal position in the competition and could certainly be improved upon further.

IV. Results

Model Evaluation and Validation

The procedures outlined in the above sections were used to obtain a final prediction score of 0.69788.

Fig 13. Final Prediction Score, lwl-rap

Justification

The Competition baseline score of 0.53792 was beaten by a considerable margin of 16%. The winning score of 0.75980 was not achieved, with a shortfall of 6%, however, the winning model and the top-scoring models documented [18] [20] [21] used training stages of 3x to 5x as long as the one implemented herein.

Additionally, the amount of pre-processing within the solution presented was negligible, afforded by the very impressive fastai2 audio library, whereas other solutions involved the significant extra time and storage usage of conversion from audio files to spectrogram images

As such, it is considered that the model performance is more than satisfactory for the task of auto-tagging audio files and further testing would be needed to see if it could achieve the same performance as the top-scorers.

V. Conclusion

Free-Form Visualization

One very clear visual element of the datasets is the difference in the noise level and quality of the recordings between the two datasets. This is shown clearly below:

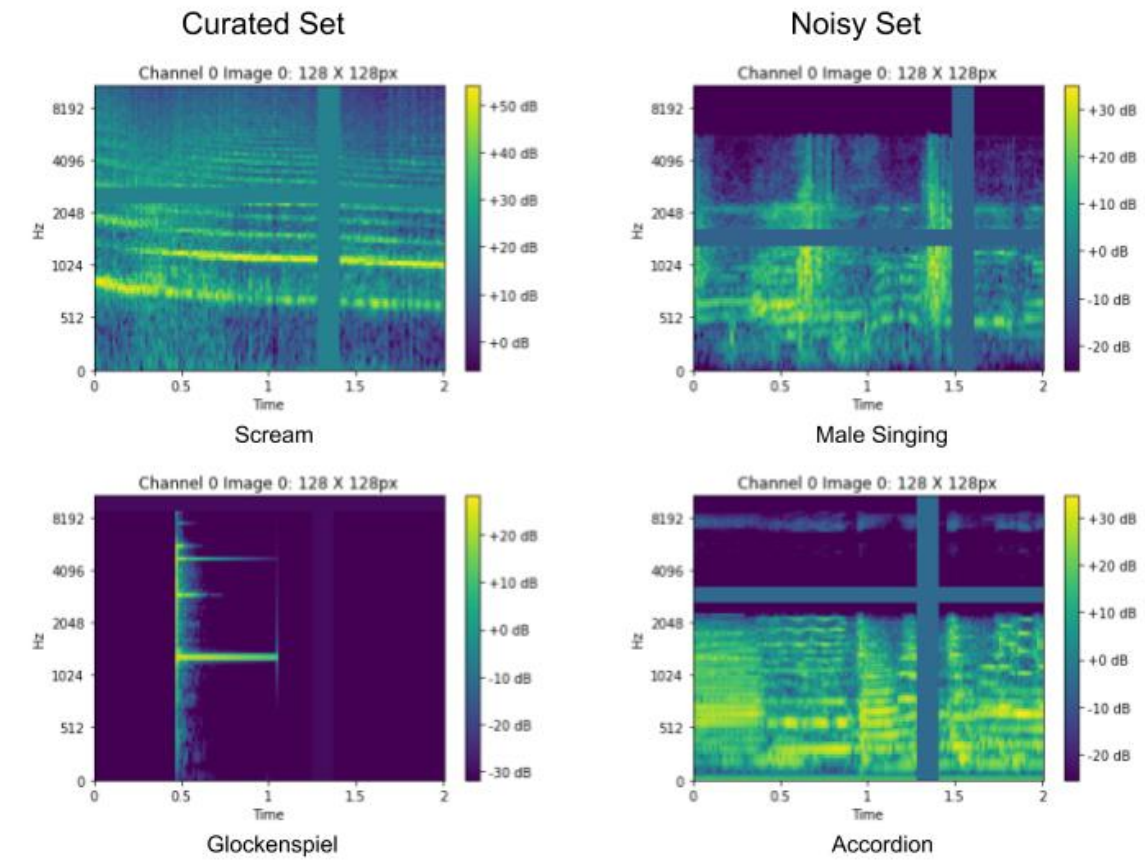

Fig 14. Differences in noise level between Curated and Noisy Set

The impact of this was that, when using the noisy dataset for training alone, the lwl-rap score was approximately 20% lower than using only the curated dataset for training. As such, the two stage training method was used for bigger performance gains.

Reflection

The stages involved in the production of this model were varied, each step requiring careful consideration of the data and audio file to spectrogram augmentations.

The most interesting aspects of the project were considered to be the time/performance cost balance required in the data augmentations' effect on the training. It was extremely interesting to also see the stark impact of the K-Folds Validation training procedure and Test-Time-Augmentations on the final score.

The main difficulties in the project were the balancing of trying to achieve a good score while minimising training and iteration costs on the Google Cloud Platform. Given the nature of the problem, the training times were necessarily long and this was not anticipated fully at the outset of choosing the problem set. In a production setting this would be a significant consideration to

bear in mind.

The final model is considered to be of good performance and will serve to inform future model development for production-stage inference of audio auto-tagging. Naturally, the TTA and ensembling of predictions means this model should only be used in an offline scenario for complex and noisy audio tagging, however, it should be noted that using this system with a standard multi-label accuracy metric, a score of over 95% was achieved within the first few epochs of training on both the Curated and Noisy Train sets.

As such, a similar approach, perhaps using a different metric (standard multi-label accuracy), could be used for a deployed inference model to be used within applications for audio classification e.g. for bird sounds, which is an area of interest of the author.

Improvement

It is considered the final prediction score could be improved in a number of ways:

- Longer training times
- Further fine-tuning of the mel-spectrogram settings
- further ensembling of models
- further data engineering, such as further training on correctly predicted noisy data [20]

The majority of the techniques were implemented within the development of the model, however, in the best performing models in the competition, more advanced data engineering was undertaken such as cross-referencing prediction scores between models and only doing further training on the audio files that were correctly selected over a certain threshold across models. This would require significant extra development that the author did not achieve within the time frame available.

---

1. http://dcase.community/challenge2020/index ↩
2. https://github.com/rbracco/fastai2_audio ↩
3. https://www.kaggle.com/c/freesound-audio-tagging-2019/overview ↩ ↩ ↩
4. Learning Sound Event Classifiers from Web Audio with Noisy Labels - Fonseca et al. 2019 https://arxiv.org/abs/1901.01189 ↩
5. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html ↩
6. https://scikit-learn.org/stable/modules/model_evaluation.html#label-ranking-average-precision) ↩
7. Fonseca et al. - *Audio tagging with noisy labels and minimal supervision*. In Proceedings of DCASE2019 Workshop, NYC, US (2019). URL: https://arxiv.org/abs/1906.02975 ↩ ↩ ↩ ↩ ↩ ↩
8. (https://colab.research.google.com/drive/1AgPdhSp7ttY18O3fEoHOQKlt_3HJDLi8) ↩
9. https://www.kaggle.com/c/deepfake-detection-challenge/discussion/129521) ↩
10. https://github.com/pandas-profiling/pandas-profiling ↩
11. Computational Analysis of Sound Scenes and Events, pg. 22 - Virtanen et al. ↩
12. https://github.com/muellerzr/Practical-Deep-Learning-for-Coders-2.0/blob/master/Computer%20Vision/04_ImageWoof.ipynb ↩
13. He, Tong, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. 2018. “Bag of Tricks for Image Classification with Convolutional Neural Networks.” *CoRR* abs/1812.01187 ↩
14. Misra, Diganta. 2019. “Mish: A Self Regularized Non-Monotonic Neural Activation Function.” ↩
15. Liyuan Liu et al. 2019 - On the Variance of the Adaptive Learning rate and Beyond ↩
16. Zhang, Lucas Hinton, Ba - Lookahead Optimizer: k steps forward, 1 step back ↩
17. Han Zhang, Ian Goodfellow, Dimitris Metaxas, Augustus Odena 2018 - Self-Attention Generative Adversarial Networks ↩
18. https://github.com/lRomul/argus-freesound ↩ ↩ ↩
19. Chan,Zhang,Chiu, Zoph,Cubuk,Le - 2019 - SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition ↩ ↩ ↩
20. https://github.com/ebouteillon/freesound-audio-tagging-2019 ↩ ↩ ↩
21. https://medium.com/@mnpinto/multi-label-audio-classification-7th-place-public-lb-solution-for-freesound-audio-tagging-2019-a7ccc0e0a02f ↩ ↩