Audio Classification of Cats and Dogs: Convolutional Neural Networks and Long Short-Term Memory Networks for deeper learning

Team 38: Chow Rui Yan, Darren Heng, Zhang Cheng Chuan

1 Introduction

Our project focuses on audio classification, particularly in differentiating animal sounds between cats and dogs. Audio classification is a machine learning task that involves the identification and labeling of audio signals. This capability allows computers to interpret the world in ways akin to humans, facilitating the automation and enhancement of tasks once deemed too complex for machines.

Despite seeming straightforward, this problem remains a hotbed for research as advancements in machine learning lead to the application of sophisticated models. These models aim for superior performance on intricate datasets such as Google's AudioSet[1], which presents challenges due to its noisy and complex nature.

We are evaluating three models: Baseline, CNN, and LSTM. Our models incorporate key AI/ML techniques, including Multilayer Perceptron Networks (MLPs), Convolutional Neural Networks (CNNs), and Long Short-Term Memory networks (LSTMs)—a type of Recurrent Neural Network. We apply concepts such as regularization and normalization consistently across all models to improve performance and generalization.

1.1 Previous Work

Our models are necessarily simpler and consequently less capable than state-of-the-art (SOTA) counterparts for a variety of reasons. SOTA models often leverage components like Transformers which require vast datasets and significant computational resources for training to achieve impressive results. Proper functioning of self-attention mechanisms in these models typically demands hundreds of millions, or even billions, of parameters[2] to be maintained in memory, in addition to substantial cloud computing costs.
In terms of architecture, popular approaches such as multi-modal and self-supervised learning present their own challenges. Our current dataset precludes the use of multi-modal models due to the absence of visual data. Moreover, the complexity and resource requirements for self-supervised learning, which often necessitates a large volume of unstructured data, were prohibitive given our constraints.
Nonetheless, we have taken inspiration from a particularly notable architecture for our CNN model. Zhang et al[3]. proposed a model that processes spectrograms as inputs and employs multiple convolutional layers to downsample the data before it reaches a fully connected layer for classification. Our adaptation of this approach aims to balance efficiency and effectiveness within the limits of our available resources.

2 Dataset

We felt that the original given dataset from Kaggle[4] was insufficient for training. It had a total of 277 .wav files of Cats and Dogs. Instead, we found another (albeit small) folder of training examples[5] that were clear and of similar lengths to the original dataset. These training examples are not noisy, with simple audio waveforms, which allows defining characteristics of the sound to be better picked up and learnt by our models.

2.1 Google Audioset

That said, we tried training on approximately 400 more examples of Cats and Dogs from Google Audioset, with

1 https://paperswithcode.com/sota/audio-classification-on-audioset

2 https://blog.research.google/2023/03/scaling-vision-transformers-to-22.html

3 http://noiselab.ucsd.edu/ECE228_2019/Reports/Report38.pdf

4 https://www.kaggle.com/datasets/mmoreaux/audio-cats-and-dogs

5 https://zenodo.org/records/3563521

Dog howls, yips, barks and Cat meows, purrs[6]. However, since the audio samples were directly taken from youtube videos, it included different kinds of sounds like car engine sounds and human conversations. The labeling was also sometimes wrong, because the .wav file would sound nothing like its label.[7]

The most important aspect however, was a fear of data shift - specifically covariate shift. Covariate shift is defined as the change in distribution of training variables compared to the testing variables. Suppose we were to use Audioset as our training data, the training data would include a lot more noise and artifacts which is very different from the clean and clear audio files in the original Cats and Dogs dataset.[8]

2.2 Exploratory Data Analysis (EDA)

For the existing two datasets, our exploratory data analysis explored basic features like amplitude-time graphs to amplitude-frequency graphs and spectrograms. Specifically, we gathered statistics like average lengths of samples, sampling rates as well as each waveform's fourier transform results.

3 Methods

The Methods section is split into 3 parts: Preprocessing, Model Implementation, Additional information.

3.1 Preprocessing and Feature Engineering

After EDA, we were convinced that feature extraction using Short-time Fourier Transform (STFT) was the most suitable way to encapsulate and embed the audio waveform into a vectorized form for more effective machine learning. Fourier Transform (FT) essentially compares different standard sine waves with the complex soundwave to produce a Frequency-Amplitude graph representing the profile of the soundwave. To capture information in the time-domain, STFT is employed, which performs FT multiple times. The resulting vectorized form of the input waveform is the spectrogram.

6 https://research.google.com/audioset/ontology/cat_1.html

7 https://github.com/audioset/ontology/blob/master/ontology.json

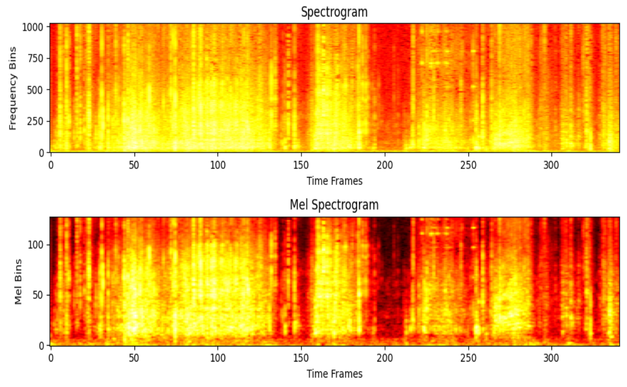
8 https://www.analyticsvidhya.com/blog/2017/07/covariate-shift-the-hidden-problem-of-real-world-data-science/ Part 4

3.1.1 Spectrograms

Spectrograms are visual representations of frequencies and amplitudes across a time period. They are 3-dimensional in that the X-axis represents time steps, Y-axis represent frequencies and the value of each cell (X, Y) represents the amplitude of that particular frequency at that time step.

3.1.2 Other variants of Spectrograms

There are also other variants of spectrograms like Mel-spectrogram and Log-Melspectrogram, with Mel-spectrogram being one of the most popular audio features. The difference between them is that the frequency bins for fourier transform are linear in normal spectrograms but logarithmic in Mel Spectrograms to more closely resemble a normal human's hearing range.[9] For our experiments, we are using only spectrograms.



3.1.3 Additional Preprocessing Steps

Since training and testing data from both sources were sampled at 22050 Hz, there was no need for further resampling. However, due to the nature of our models, we had to ensure a uniform length for the spectrogram. To enforce this, we truncated waveforms of greater than 200,000 samples and padded those below that threshold, with the value being empirically determined in our EDA.

3.2 Model Implementation

We implemented 3 models: Baseline (MLP), CNN and RNN (LSTM)

9 https://essentia.upf.edu/tutorial_spectral_representations.html

3.2.1 Multilayer Perceptron (Baseline)

This was developed as a basis of comparison to the other models. It is a Multilayer perceptron neural network with 2 Hidden layers and 2 output nodes, with dropout layers in between and LeakyReLU. As the spectrogram contains a lot of information (3 Dimensions), flattening it into a column vector MLP will lead to a lot of input nodes. Specifically, there are 200900 input nodes according to our pre-set FFT parameters.

3.2.2 Convolutional Neural Network

This was one of the two main models we wanted to implement. This is a classic CNN and MLP set up, with 4 convolutional and pooling layers and a fully connected neural network at the end. The input spectrograms undergo convolutions, batch normalization, LeakyReLU, dropout and max pooling.

CNNs are good in capturing local spatial features within the spectrogram, identifying patterns and "textures". It also serves to downsample and reduce the dimensions of inputs for classification in the fully connected layers.

3.2.3 Long Short Term Memory Network

This is the other model, which is a LSTM network similar to the CNN model above, except that instead of CNN, we use a LSTM network with multiple hidden layers. LSTMs are better than RNNs due to a number of reasons[10], but most importantly, like the RNNs, LSTMs can keep track of and learn temporal features and dependencies which is absolutely crucial when classifying sounds, which can have different frequencies and amplitudes over a period of time. This is done through the many hidden states that are folded deep within the LSTM.

For our project, we used a bidirectional LSTM because we receive as input the entire .wav file and there is no need to classify unidirectionally like in real-time.

3.3 Additional Steps

We employed Xavier Initialization instead of randomly initialized weights for better training. The intuition behind Xavier Initialization is like drawing randomly from a normal distribution rather than uniform distribution. With these weights, the activations afterwards will have the same variance. This reduces the impact of exploding or vanishing gradients.[11]

Batch Normalization was employed as well in LSTM and CNN. This has the effect of standardizing the entire training batch, much like data "whitening" in computer vision. This stabilizes and speeds up training, with the assumption that the data distribution before the weights update is similar to that of after the update.[12]

4 Results and Discussions

For the majority of comparisons, we use F1, F2 scores, as well as Mean Average Precision (mAP) as our metrics. We focus more on mAP because that is the most commonly used benchmark for audio classification of Google Audioset[13].

Average Precision can be seen as the average value of precision at each recall level for each label. Averaging this across all labels will give the mAP. mAP is good at giving a clearer view of performance of the model, especially in cases of label imbalance.

We had 3 parameters with a total of 12 total permutation of variables for each model. With 3 models, we ran a total of 36 experiments with the finalized models.

4.1 Absolute Performance

The F1, F2 and mAP values are averaged across all the different hyperparameters (12 permutations per model)

Model/ Metrics	F1	F2	mAP
Baseline	0.84713883	0.82419832	0.81359601
CNN	0.71057151	0.63718609	0.73322368
LSTM	0.75438632	0.7272228	0.71452496

Previously, we thought that the performance of baseline will be the poorest because MLP networks are not good at learning the structures of the spectrogram and identify spatial and temporal features within. However we were

10 https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/

11 https://cs230.stanford.edu/section/4/

12 https://machinelearningmastery.com/batch-normalization-for-training-of-deep-neural-networks/

13 https://paperswithcode.com/sota/audio-classification-on-audioset

surprised that the baseline model significantly outperforms the other two models, be it in F1, F2 or mAP scores.

We attributed this phenomenon to a few factors. Firstly, we believe that the size of datasets and samples matter. Our training data had about 280 samples of both cats and dogs. Each sample was either padded or truncated to about 8 seconds long. Since the variance of the training sample length was large, many audio clips were incomplete or padded with all zeroes. This can lead to situations where the actual bark or meow of an audio sample was cut-off or removed completely, or "diluted" by the zeroes.

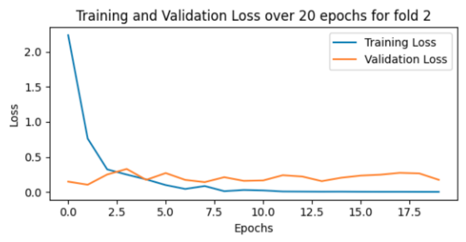
Another reason could be that we were unable to properly optimize the hyperparameters in the complex models CNN and LSTM. Our hyperparameters were only "generic" hyperparameters, while other model-specific ones were not tested due to lack of time and compute resources. For our experiments, the number of frequency bins (n_fft) and time steps (hop_length) remained constant at 2048 and 1024, which meant that an important aspect of the CNN was not tuned. It has been shown that greater number for n_fft can lead to much better performance[14]. For both LSTM and CNNs, the number of hidden_states or number of channels were also not hyper optimized

The last reason could be due to the length of training. Each CNN model took approximately 75 minutes to train for 20 epochs and 4 KFold Cross Validation steps. LSTM took approximately 30 minutes while Baseline models took about 15 minutes. Should time permit, we could increase the number of epochs to allow the more complicated models to learn better.

4.2 Data Visualization

4.2.1 Epoch-Loss Graph (ELG)

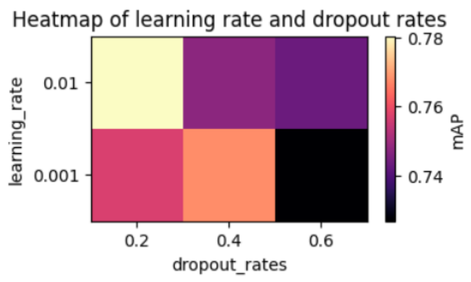
Most of our ELGs follow conventional ELGs in that the Training and Validation Losses decrease with epochs. That said, there are instances where abnormalities occur, such as non-decreasing validation losses. Common points to note are that some Validation Losses go up after a period of decrease due to overtraining or overfitting.



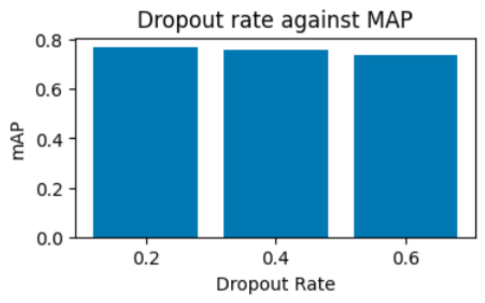
[14] https://arxiv.org/pdf/2001.09989.pdf

For example, in this instance of ELG, the validation loss has not started to rise, so we postulate that the optimal number of epochs for training is greater than the number we set.

4.2.2 Hyperparameter-pair Heatmap and Bar Charts



To better understand the relationship between different pairs of hyperparameters, we created heatmaps to illustrate their impact on the overall average mAP of different models.



4.3 Insights and Conclusion

For the hyperparameters, we observe that the optimal dropout rate is 0.2, learning rate is 0.01 while the difference between batch size 64 and 128 is not clearly shown.

As for the model architecture, we still believe that LSTMs and CNNs are better and look forward to allocating more resources to it to find out more. But for optimal performance, we tried using transfer learning from VGGNet and ResNets with spectrograms as input in our preliminary testing but decided against it. We wanted to try implementing and iterating on our own models for better learning.

To sum all the findings up, we believe that good model architectures also need resources to do well, be it high quality data or great amounts of compute resources. We look forward to repeating our experiments on a larger scale in the future.