PROJECT REPORT

Efficient Speech Emotion Classification Using Transfer Learning

Shane Berhoff and Andrew Lu

Emory University College of Arts and Sciences

Correspondence: shane.berhoff@emory.edu; andrew.lu@emory.edu

Abstract

The recent growth of multi-modal agents has paralleled the growing desire for effective on-device personal assistants. This presents a need for efficient speech emotion classification to give these agents greater situational understanding and more humanized responses. To achieve this we broke with the norm of text sentiment analysis and opted for a pure audio approach to fully capture the nuance of human speech. Previous research has primarily relied on some form of text transcription and/or MFCCs in addition to computationally expensive architecture not transferable to an on-device setting. We hypothesised that improved accuracy could be gained through a pure audio Mel Spectrogram representation approach, due to a larger information retention of the original audio data, while also using relatively lightweight architecture. Through methods of precise data curation, transfer learning, and parameter tuning a maximum accuracy of 78.62% was achieved on a 6 class problem through Inception V3 Half Cycle. Additionally an accuracy of 75% was achieved on MobileNet V3 which proved to be over 3 times faster at inference time and is specifically built and optimized for mobile architecture. Overall the indication is that this pure audio approach could prove useful for future on-device, human speech, and multi-modal agent related applications.

Keywords: transfer learning; neural networks; machine learning; cnn; speech

Abbreviations: WA: Weighted Accuracy UA: Unweighted Accuracy

1. Introduction

With the increasing usage of LLMs such as ChatGPT, Claude, and Gemini, pushing the accuracy of responses has been a growing field of research, with metrics such as MMLU (Undergraduate level knowledge) and HellaSwag (Common knowledge) [1] driving research. However, in the case of personal on-device assistants such as Siri and Alexa, which either use smaller LLMs or a robust command processing system behind the scenes, or more generally multi-modal models that also incorporate speech into their inputs, pure Automatic Speech Recognition methods for transcription of speech may not provide the most accurate results.

Take, for example, the short context windows of most interactions with Siri or Alexa. Most interactions revolve around a few short sentences, and without a larger context of dialogue, it is extremely difficult for LLMs to properly capture emotional nuance from purely text itself. Because of this, capturing nuance in speech that is lost when transcribed to text is important as researchers and engineers try to provide a more accurate and natural way to interact with these digital assistants.

Our Machine Learning project seeks to address this issue by utilizing lightweight models trained purely on labeled speech in order to provide a feasible, on-device solution for assistants such as Siri or Alexa to make the most out of their short interaction windows by also capturing the emotion behind varying utterances in speech.

2. Background

In our initial research, we found that the majority of existing studies employed methods for emotion classification that closely mirrored those used in text transcription tasks.

For example, the first paper we found was written by Lu et al. [2], which utilized the "black box" vector output of the encoder weights from the middle of an Automatic Speech Recognition task (used for text transcription) as a feature extractor which was then fed into a pre-trained Recurrent Neural Network Transducer model (with Long-Short Term Memory). This approach made sense, as the authors hypothesized that the encoder output properly represented features relevant to capturing acoustic characteristics, and RNN-Ts are useful for temporal data. The authors reported 71.7% weighted accuracy and 72.6% unweighted accuracy on a different dataset with 4 classes and approximately 10,000 utterances.

The second paper we found was written by Liu et al. [3] and utilized an LSTM + Convolution network as a feature weigher and Wav2Vec2 as a feature extractor to predict 4 classes with approximately 5500 utterances, with the hypothesis that by weighing various features differently and combining both Wav2Vec2's word detection with auditory characteristics, performance could be improved. Overall, the authors reported 73.18% weighted accuracy and 74.26% unweighted accuracy.

Though performance across the board was impressive, we noted that both papers utilized some part of the text transcription pipeline for speech emotion recognition, and that both proposed architectures were computationally expensive - both LSTMs and Transformer models such as Wav2Vec2 tend to be more expensive to run and hold in memory, something not conducive to performance on smaller devices. Additionally, both models utilized Mel-Frequency Cepstral Coefficients, which are highly compressed from the original Mel Spectrogram, and despite their usefulness for speaker diarization and speech detection lose much of the spoken nuance necessary for accurate emotion analysis.

Therefore, we chose to utilize Mel Spectrograms for conveying audio information accurately while preserving fine features and eliminating the need for text transcription. Additionally, we use convolutional neural networks in place of recursive neural networks to develop a less computationally demanding model.

3. Methods

3.1 Metadata & Statistics

The chosen dataset [4] is a grouping of 4 popular datasets in the area of human speech compiled on Kaggle: Crema, Savee, Tess, and Ravee. This produces the need for initial data cleaning as there are slight differences in formatting and labeling of samples between the sub-datasets. We wrote a metadata script to search through all data files where it limits to classes contained in all subsets, and creates a master DataFrame that stores the location of each sample and its corresponding class label. There were 6 classes used: sad, angry, disgust, fear, happy, and neutral coded 0-5 respectively.

Then taking these samples we determined useful statistics for the purpose of standardizing the data. There were 11318 total samples used with an average audio clip length of 2618 milliseconds (2.618s). All samples were padded (using reflection) or truncated to this length and stripped down to single channel. The distribution of sample rates among samples can be seen in Figure 1. We chose to re-scale all samples to 24414 Hz in an attempt to minimize both information loss and up-scaling. This up-scales samples at 16000 Hz and down-scales samples at 48000 Hz, 44100 Hz and 96000 Hz. This re-scaling of the data came with a large time cost so it was done as a preliminary step after which samples were stored as tensors. This time cost would not be incurred in a production real

time application given that audio is recorded at the same sample rate that the model is trained on.

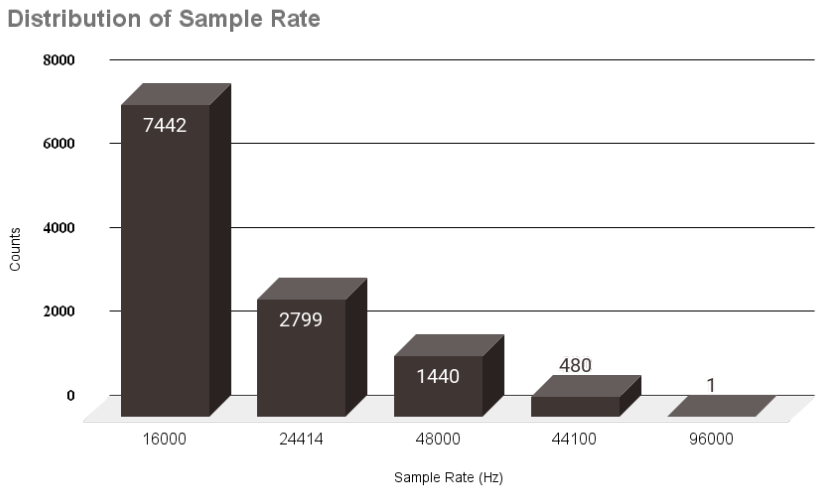

Figure 1. Sample rate distribution across all four datasets.

3.2 Data Pipeline

There are some train time adaptations that can be made to our data that assist in reducing over fitting and helping overall model generalization. First we load the tensor and class from system storage using our previously produced metadata. Then the audio is time-shifted (with wrapping) a random amount by up to 30% of its length. After this the sample is converted to a Mel Spectrogram. The reasoning behind the specific number of Mel bands, length of FFT window, and hop length for this conversion will be discussed more in depth later on in section 3.4.2. Finally the produced spectrograms are augmented with 2 masks in the frequency domain and 2 masks in the time domain of a random size with max width being 10% of the domain. A visualization for what a sample looks like coming out of the pipeline can be seen in Figure 2.

We created custom PyTorch DataLoaders that perform these actions for each epoch during the training process and injects the samples in batches. This creates differences in the training data between epochs and improves model generalization.

3.3 Homebrew Model Architecture

To test our data pipeline process and the validity of our idea we created a basic homebrew model. This is a convolutional neural network made up of 4 convolution blocks. Each block consists of a 2D convolution layer with the Rectified Linear Unit activation function ($f(x) = \max(0, x)$) applied to its output. The outputs of the activation function then undergo batch normalization to stabilize the learning process. This block is initialized using Kaiming initialization which is effective for layers followed by ReLU activation. The 4 convolution blocks output 8, 16, 32, and 64 feature maps respectivly which are then fed into an adaptive averaging pooling layer to stabilize the dimension of inputs and the final fully connected linear layer that takes in 64 features and outputs classification on the 6 classes.

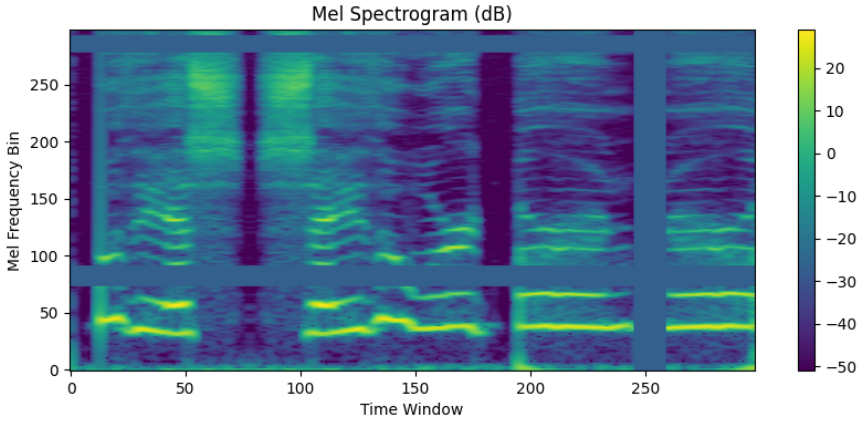

Figure 2. An example of data processed into a spectrogram right before it is fed into a model. The blue bars across the image are the time and frequency masks from the augmentation step.

3.4 Transfer Learning

3.4.1 Overview

In additional to building our own homebrew model, we chose to utilize transfer learning because of the success of learning generalization to medical imaging using models pretrained on the ImageNet dataset [5], despite the tremendous visual differences in tasks. We hypothesize that the lower-level features learned on ImageNet are still applicable to spectrogram classification. Additionally, with transfer learning, we can overcome the problem of our smaller dataset and data scarcity, allowing us to utilize slightly larger models and potentially boost accuracy.

Out of all the models with pre-trained weights we found, we chose four in particular to perform transfer learning: MobileNet V3 Large, EfficientNet V2-B0, EfficientNet V2-B1, and Inception V3 for the variety of architectures, sizes, and predict times between each model.

3.4.2 Dimension Adjustment

When utilizing pre-trained models with existing architecture we need to make sure that our samples match up with the required input dimensions of each model. We could simply transform the data to the specific dimension at the end of our data pipeline but this is not the best way to go about it. In case 1: if the generated spectrogram is to large and needs to be cut down in dimension then information is lost. In case 2: if the generated spectrogram is to small in dimension and it needs to be scaled up, data was not lost in this step but max information was not achieved. This is because spectrogram conversion is inherently a lossy operation, so by making a spectrogram that was too small information was lost. Therefore the best way to achieve maximum information retention and correct dimensions is to adjust our existing DataLoaders to generate spectrograms of perfect dimension for each model specifically.

The final dimensions of a spectrogram depend on:

- Sample rate of audio
- Duration of audio
- Number of Mel bands
- Length of Fast Fourier transform window
- Hop length

In this case the sample rate of audio is always going to be 24414 Hz due to the resampling procedure, and the duration was also previously standardized to 2.618 seconds. This is where specifications for the specific number of Mel bands, length of FFT window, and hop length (as mentioned in section 3.2) come in. The y (frequency) dimension is directly determined by the number of Mel bands. So if Mel bands is set to 224 then the y dimension will be 224. The x (time) dimension is more complicated with a general formula of:

$$x\_frames = \frac{(sample\_rate)(duration) - n\_fft}{hop\_len} + 1 \tag{1}$$

Given the constant values of sample rate and duration we also chose to fix $n\_fft = 2048$ between all models and samples. This leaves a sole dependence on hop length where Eq. 1 can be rearranged to produce Eq. 2 with hop_len in terms of required number of frames and constants.

$$hop\_len = \frac{(sample\_rate)(duration) - n\_fft}{x\_frames - 1} \tag{2}$$

This provides a very close approximation for a hop length (in conjunction with all other values) to produce the correct overall dimensions. However, it is not perfect because of the potential for partial frames at the beginning or end of audio clips, frame overlap, and the requirement for hop length to be an integer. Some iterative adjustment is required moving the hop length up or down slightly untill dimensions are correct. For our data specifically we experimentally determined a slightly more accurate formula for hop length to produce closer to correct final x dimensions:

$$hop\_len = \left\lceil \frac{(sample\_rate)(duration) - n\_fft}{x\_frames - 1} \right\rceil + 3 \tag{3}$$

Now our DataLoaders use these new calculations to produce spectrograms of near perfect dimension for each model specially to provide maximum information retention going into a final transform. This final transform ensures that they are the correct dimension (adjusting slightly if needed), converts from 1 channel to a 3 channel pseudo-RGB representation that the image model is expecting, and normalizes it according to the data the model was originally trained on. This process allows our data to be fed directly into the pre-trained models at train time.

3.4.3 MobileNet V3 Large

The first model we chose was MobileNet V3 Large - this model was designed for efficient inference on mobile devices, with approximately 5.4 million parameters and 440 million FLOPs per inference. In addition to a lightweight classification head, MobileNet V3 utilizes an even more efficient activation function (Hard Swish) and some unique blocks such as Bottleneck (efficient feature extraction) and Squeeze-and-Excitation (weights for each feature map) in addition to a full network search for better performance to size. We hypothesize that the transfer-learning MobileNet V3 model will perform the worst out of all our options due to its smaller size.

3.4.4 EfficientNet V2-B0/1

The second and third models we chose were EfficientNet V2-B0 and V2-B1, with approximately 7.4 million/8.1 million parameters and 700 million/1.2 billion FLOPs per inference. Both models were also designed for high peformance/efficiency. Both also have lightweight classification heads in addition to adaptive regularization and Fused-MBConv blocks (combining both Bottleneck and Convolution) for more efficient convolutions. We hypothesize that both EfficientNet models will perform better than MobileNet as it should be able to capture more details in the spectrograms but slightly worse than Inception.

3.4.5 Inception V3

The fourth and final model we chose waes we determined useful statistics for the purpose of standardizing the data. There were 11318 total samples used with an average audio clip length of 2618 milliseconds (2.618s). All samples were padded (using reflection) or truncated to this length and stripped down to single channel. The distribution of sample rates among samples can be seen in Figure 1. We chose to re-scale all samples to 24414 Hz in an attempt to minimize both information loss and up-scaling. This up-scales samples at 16000 Hz and down-s Inception V3. Compared to the other models, Inception is significantly larger with approximately 25 million parameters and 6 billion FLOPs per inference. Aside from having asymmetric convolutions ($1\times n$ and $n\times 1$ blocks) in addition to factorized convolutions (breaking down larger convolutions into a chain of smaller convolutions), Inception V3 utilizes Inception Modules that consist of multiple parallel convolution branches to learn a variety of features at each level. We hypothesize that Inception V3 will outperform all our transfer learning models and potentially break past prior research's performance.

3.4.6 Adjustments

Finally, in order to classify to our 6 classes, we dropped the final linear layer with width 1000 in each model and replaced it with a new linear layer with width six in order to predict our emotion classes properly. We did some preliminary testing by adding some more custom linear layers on top of the initial predictive layer, but found that the train time complexity in addition to our sparse data resulted in poor performance compared to a singular linear layer.

For Inception V3, we chose to freeze all layers except the final 8 mixed layers as specified in timm's Inception V3 implementation, namely the 7x7 branches and final 3x3 branches in order to cut down train times for Inception V3 in addition to reducing overfitting due to the sheer number of parameters.

4. Experiment/Results

4.1 Project Setup

Initially, we attempted to do resampling, rechanneling, and audio duration standardization in addition to our other augmentations at runtime (as mentioned in 3.1 and 3.2). However, after realizing CUDA usage was virtually undetectable at runtime, we chose to perform resampling, rechanneling, and length standardization beforehand and saving our resampled, rechanneled, and data as tensors, cutting the train time of one epoch from around 40 minutes to approximately 30 seconds. Augmentations in our pipeline involving masking, shifting, and spectrogram generation were negligible and were not a bottleneck given enough workers, and would also be negligible on smaller devices performing inference on a sequence of individual samples.

Additionally, within our DataLoaders, we implemented train/test split functionality in order to generate a list of samples used for training/testing for each individual models in addition to no augmentation to test data in order to prevent data leakage and provide deterministic test accuracy results for the purpose of reporting results.

4.2 Parameter Selection & Training

Once we overcame our training bottlenecks, we did a training run through each model with a max epoch of 100 and early stopping with patience 5 utilizing the Adam optimizer with a learning rate of 0.001 and a One Cycle Learning Rate scheduler with a max learning rate of 0.001 and linear annealing, essentially skipping the "warm-up" phase of One Cycle Learning Rate to determine a reasonable epoch count before convergence for each model. For the rest of this report, we refer to skipping the warm-up phase of One Cycle LR as a "Half Cycle Learning Rate."

We chose this optimizer and scheduler because they were mentioned in prior computer vision articles we read for reference, and the Adam optimizer would be good for high-dimensional parameter spaces; when combined with the One Cycle LR scheduler to cycle through a higher and lower learning rate once, this combination would help us explore the loss landscape better and hopefully attain better results.

Once we obtained a reasonable epoch count, we then created our own homebrew model in addition to utilized timm's pre-trained models, providing transformations and adjustments in each as defined in Section 3.4.6 for models defined in Section 3.4. For each model, we used the approximate max epoch determined above and utilized two different optimizer/scheduler pairs (Adam with 0.001 learning rate and Half Cycle LR and Adam with 0.0001 learning rate and One Cycle LR with 0.001 max learning rate). Finally, we create a simple pipeline to sequentially train and collect data from all five models using the two optimizer/scheduler pairs automatically to allow for overnight training.

4.3 Results

After compiling train results, we obtain the following test accuracies, with Inception V3 Half Cycle performing the best, followed by MobileNet V3 and EfficientNet V2-B0:

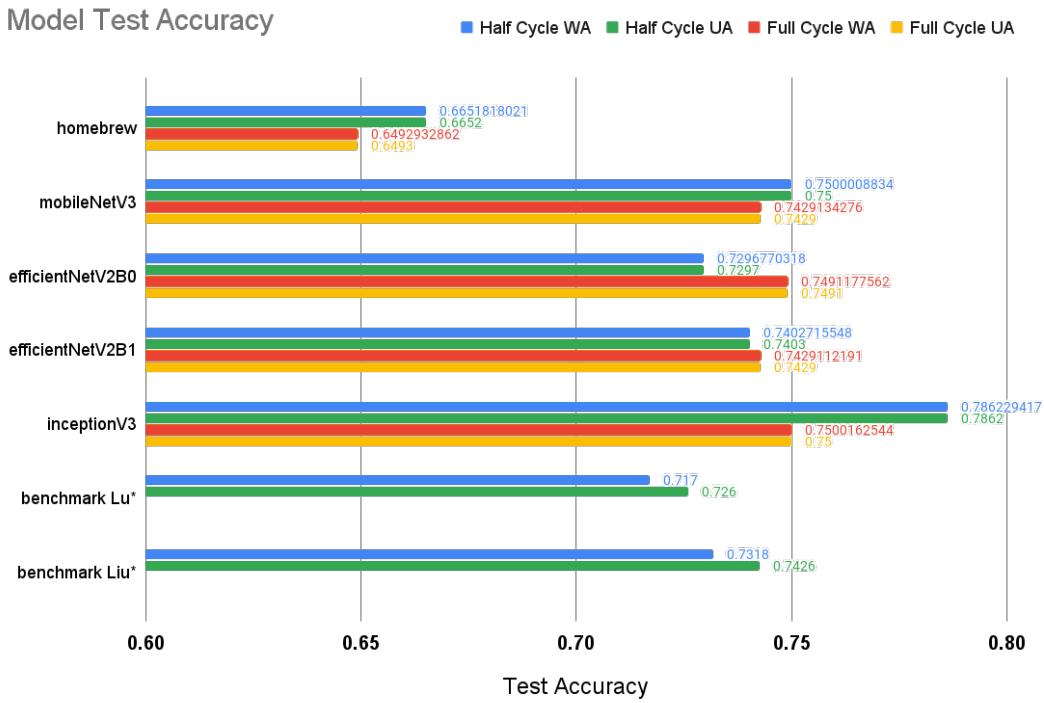

Figure 3. Overall model test accuracy. For benchmark models, the top bar is reported WA and the bottom bar is reported UA.

Additionally, we compiled accuracy for each emotion, performing averaging and comparisons across emotions:

Table 1. Emotion-wise accuracy, averaged for both half- and full-cycle scheduling for each model.

Emotion	homebrew	mobileNetV3	efficientNetV2B0	efficientNetV2B1	inceptionV3
Sad	0.6886	0.71245	0.7162	0.68295	0.72305
Angry	0.8101	0.8826	0.84005	0.85535	0.82495
Disgust	0.59875	0.64665	0.6174	0.7085	0.76895
Fear	0.48135	0.69245	0.68895	0.6457	0.68625
Happy	0.6228	0.6829	0.71655	0.815	0.7601
Neutral	0.77695	0.8598	0.86565	0.75025	0.8531

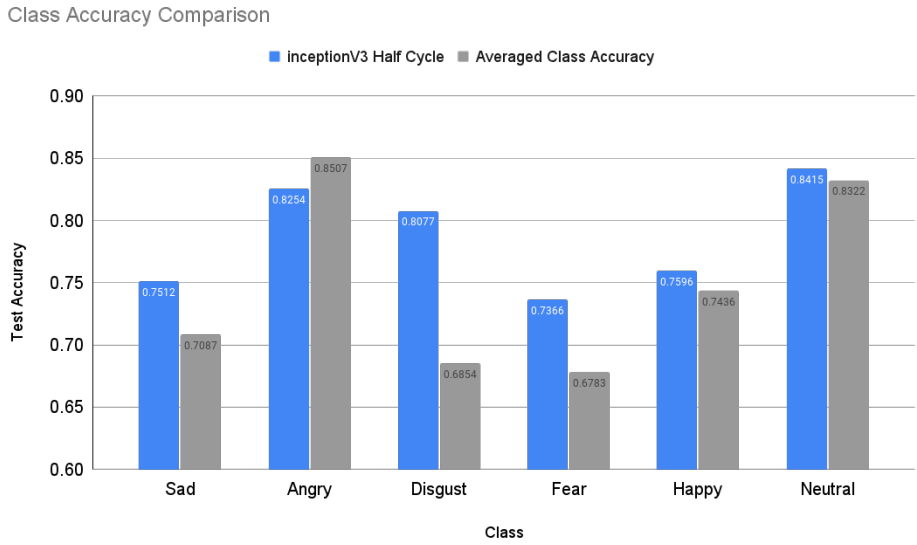

Figure 4. Average emotion-wise accuracy across transfer learning models compared to the best performing model (Inception V3 Half-Cycle).

We also ran inference on each model on a laptop with an AMD Ryzen 7 5800U CPU @1.9 GHz with batch size one and one worker, calculating the average inference time over five runs to see the relative inference time between models.

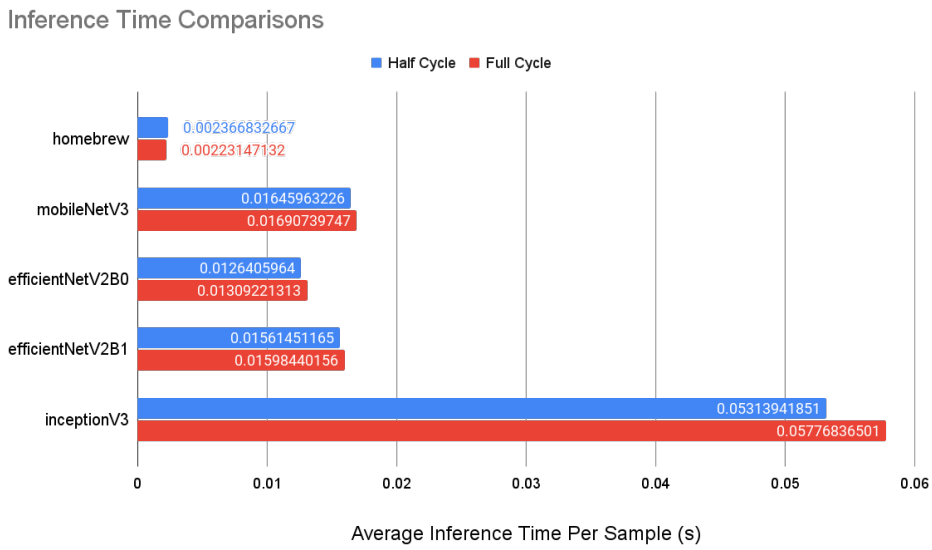

Figure 5. Caption

5. Discussions

5.1 Overall Accuracy

Each of our transfer learning models has at least one learning rate/scheduler combination that outperforms both benchmark models - additionally, Inception V3 trained on the Half Cycle Scheduler absolutely dominates the other models' performances, with second place being Inception V3 trained on One Cycle LR and third place being MobileNet V3 Large trained on Half Cycle LR. Inception V3 Half Cycle beats Lu et al.'s model ([2]) by 6.92% and Liu et al.'s model ([3]) by 5.44% in unweighted accuracy and 6.02% and 4.36% weighted accuracy respectively (Figure 3).

Unfortunately, the two papers did not report relative runtimes or model size, so we are unable to draw a conclusion at this point regarding performance to model complexity. Additionally, because the two papers utilized different datasets, our performance is not directly translatable to their dataset - however, it is worth noting that our models had to predict between two additional classes, and future work will involve also training our models on the IEMOCAP dataset used by both Lu et al. and Liu et al. This can be done through transfer learning from the initial ImageNet weights or tuned using our models' current weights, as more spectrogram specific features have been learned from our dataset.

5.2 Emotion-wise Accuracy

When looking at accuracy for each emotion separately, it seems that for our chosen speech datasets that sadness, disgust, and fear are the most difficult classes to predict with an average accuracy of 70.87%, 68.54%, and 67.83% accuracy respectively, with anger, happiness, and neutral speech the easiest to classify with 85.07%, 74.36%, and 83.22% average accuracy respectively (Figure 4). Doing a very meta analysis of speech and listening back on some samples from within our dataset, this does make sense - for example, anger has a very distinct, forceful tone across voice actors, whereas the tonal properties of fearful utterances are much more muted in comparison. Interestingly enough, our top performing model, Inception V3 Half Cycle, actually breaks this trend as seen in Figure 4,

with disgust being classified more accurately than happiness - we suspect this may have to do in part with how our optimizer and scheduler found a good minima on the loss landscape in combination to a good train-test split that allowed the model to generalize well across classes on the data it was given.

5.3 Wide Performance Range

However, when comparing our transfer learning models to each other, it is interesting to note that MobileNet V3 Large Half Cycle does almost just as well as Inception V3 Full Cycle (>0.002% difference in performance, Figure 3) while taking 70.73% less time on inference per sample (Figure 5 - additionally, the discrepancies in performance between the same models on different schedulers seems to grow very high (upwards of 3.62% for Inception V3).

Since the transfer learning models were initialized to the same weights regardless of scheduling method, we suspect that this discrepancy may be in a large part due to the train-test split generated, which is different from model to model. Additionally, because each of the transfer learning models incorporates dropout to one extent or another, we also think that this wide range of performance can be due to what pathways are de-emphasized randomly during train time. To address this in future project modifications, we intend to use k-fold validation to smooth the discrepancies from run to run by averaging the variability in the initial train-test split of the data.

5.4 Conclusion

Overall, our results are promising in demonstrating the capability of lighter convolutional neural networks with a variety of architectures on multi-emotion classification. Though we beat prior research by a significant margin, we note the very promising performance of smaller models such as MobileNet V3, which is almost 4 times faster than our best performing model.

Additionally, this project more generally demonstrates the promising capabilities of transfer learning from models pretrained on ImageNet despite the very drastic visual differences in tasks - lower-level features learned from ImageNet have been demonstrated to perform well on medical imaging, and this project also demonstrates good generalization to spectrogram classification.

6. Contributions

Shane Berhoff was responsible for the majority of pre-preprocessing (such as metadata collection, aggregation, and the conversion of audio files to tensors) in addition to later tasks such as DataLoader creation, spectrogram conversion, solving dimensionality discrepancies, and test architecture (homebrew model). Andrew Lu was responsible for the majority of transfer learning (such as creating train/predict scripts, modifying predefined models, and model data collection) in addition to training tasks such as building pipelines, collecting model performance data, and selecting hyperparameters.

Though we focused on different areas of the project, we both contributed to all aspects of the project both in code and in processing power depending on how our devices allowed (Shane could generate tensors significantly faster than due to his M1 chip, but Andrew could train models substantially faster due to his laptop's NVIDIA GPU) - feel free to look at our GitHub contributions in the following section. We both wrote presentation slides and the report sections for our respective tasks and split graph generation and proofreading between us.

7. Code

Due to the size of our dataset and pre-trained model weights, we separated the source code from our data and outputs. Please reach out if you have problems downloading or using our tensors or models.

- Github: CS334 Project
- Original Dataset: Kaggle
- Converted Tensors: Onedrive tar.gz
- Models, Pretrained Weights, Training Logs, and Train-Test Splits: Onedrive tar.gz

References

[1] Anthropic AI. *Introducing the next generation of Claude*. Tech. rep. 2024. URL: https://www.anthropic.com/news/claude-3-family.

[2] Zhiyun Lu, Liangliang Cao, Yu Zhang, Chung-Cheng Chiu, and James Fan. "Speech sentiment analysis via pre-trained features from end-to-end ASR models". In: *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing* (2020). DOI: 10.1109/icassp40776.2020.9052937.

[3] Ke Liu, Dekui Wang, and Jun Feng. "Speech emotion recognition via two-stream pooling attention with discriminative channel weighting". In: *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech, and Signal Processing* (2023). DOI: 10.1109/icassp49357.2023.10095588.

[4] *Speech Emotion Recognition*. https://www.kaggle.com/datasets/dmitrybabko/speech-emotion-recognition-en. [Online; accessed 20-March-2024].

[5] Hee Kim, Alejandro Cosa-Linan, Nandhini Santhanam, Mahboubeh Jannesari, Mate E. Maros, and Thomas Ganslandt. "Transfer learning for medical image classification: a literature review". In: *BMC Med Imaging* 22 (2022). DOI: 10.1186/s12880-022-00793-7.