ELEC 301 Final Project Report

Bill Nguyen, Jack Pearce

PROBLEM STATEMENT

The task this project addresses is a classification task given labeled training data; to accurately classify unlabeled (actor) speech data (audio) as being 1 of 8 emotions:

- angry, sad, calm, happy, fearful, disgust, surprised, neutral

Success was measured primarily by the accuracy score a given model could achieve once submitted to the Kaggle leaderboard, as said score was a test of predictive power on data that we truly could not access (and thus there was no chance of that metric being inaccurate due to data leakage.)

A secondary measure of success we focused on was the testing of new, relatively unexplored methods and techniques, especially when relating to more powerful features or faster/more efficient approaches to achieve similar results to, say, relatively slow CNNs. This goal was placed both because of our resource limitations, our data limitations, and the desire to achieve something new and inventive in the project (in an area that may be relatively unexplored.)

Constraints

Most emotions within the training dataset were on the order of 100-200 samples, meaning that effective prediction could not rely on an extremely high *n*. Thankfully, audio clips are themselves very information-laden; most models in our project worked off of unwrapping audio clips into information-dense features that generally sufficed for our purposes.

Additionally, a certain amount of GPU/Memory usage would crash Colab; thus, we were forced to cap the amount of GPU usage, to the amount that about 4 CNN convolutions would take.

Here are our constraints, listed in tl;dr format:

- Jeff's deadlines
- Colab usage limits and CPU/GPU RAM
- Our imaginaaaation (and willingness to learn/implement stuff)

Data Exploration

The actors are informed to intentionally convey one of these emotions per audio clip, and all clips are one of two sentences ("dogs are sitting by the door" or "kids are talking by the door"). Both female and male actors are included in the clips.

Characteristics:

- **Imbalanced data:** Neutral emotion has half as many clips as the other emotions. (This turned out to not be a problem though, as SVMs can use balancing class weights, and Neural Networks learned well despite the imbalance)
- **Class overlap:** Some emotions such as anger/disgust can be expressed in different ways, and some emotions sound very similar (sad/calm/neutral).
- **Gendered expression**: Men and women noticeably express certain emotions differently
- **Sentence semantics**: There were only 2 sentences being spoken.
- **Audio format**: Some were mono and some were stereo, and they all had different lengths.

Exploration:

- **Dimensionality:** Longest audio clip was 253,053 samples long → difficult to classify raw. Thus, we tested primarily with MFCC-like features as a way to operate on information-efficient dimension-reduced data.
- **Clustering**: Unrolled MFCCs not visually separable with PCA (at 2-3) dimensions, but visually separable with t-SNE. Upon testing with SVMs we had found that t-SNE data was 67% accurate (on training data) on a linear SVM, but this accuracy was not adequate, given the dataset it was evaluated on. SVM on PCA showed a significant reduction in accuracy compared to unrolled MFCC.

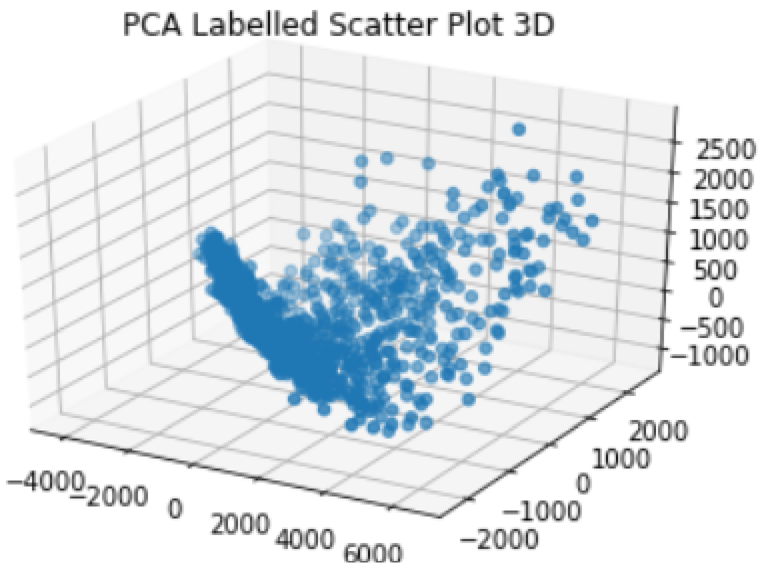


Figure 1: 3D PCA Plot. This is of MFCC's and pre-augmentation.

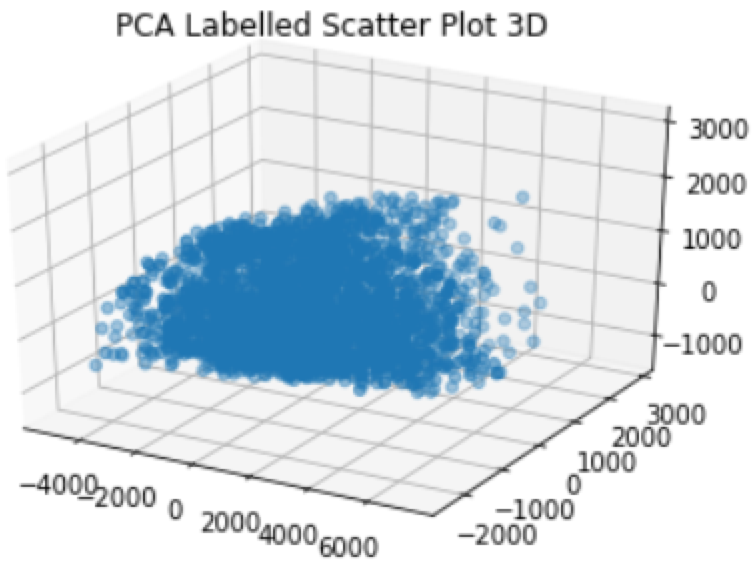


Figure 2: 3D PCA on augmented audio MFCC's. Post-augmentation the hammock shape becomes a tube. Despite having shape, there is not a clear visual separation between clusters.

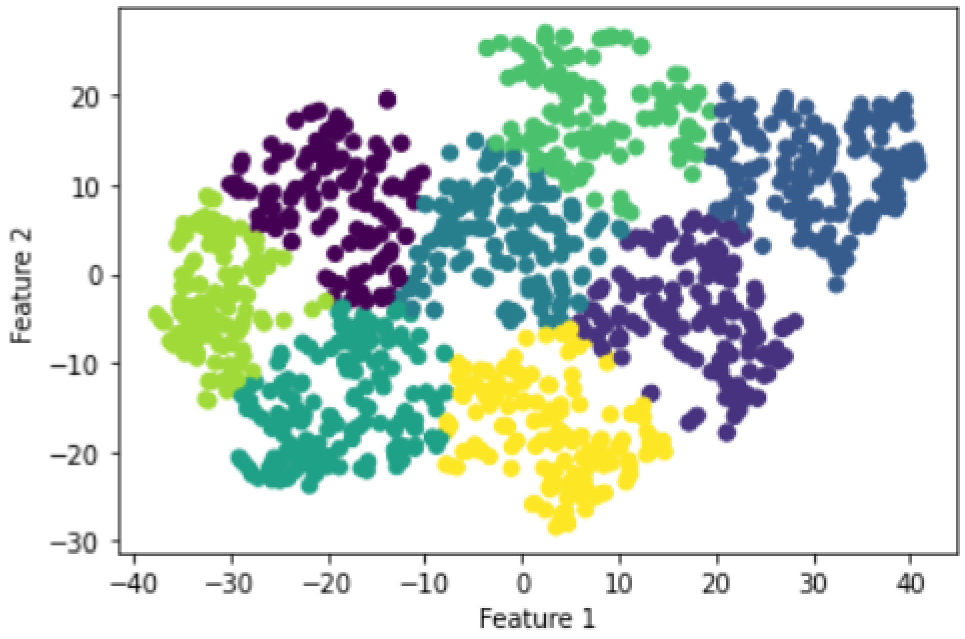


Figure 3: K-Means clustering run on t-SNE dimensionality reduction of unaugmented MFCCs.

- **FFT/Audio plotting** yielded no clear and immediate visual/numerical difference between emotions.

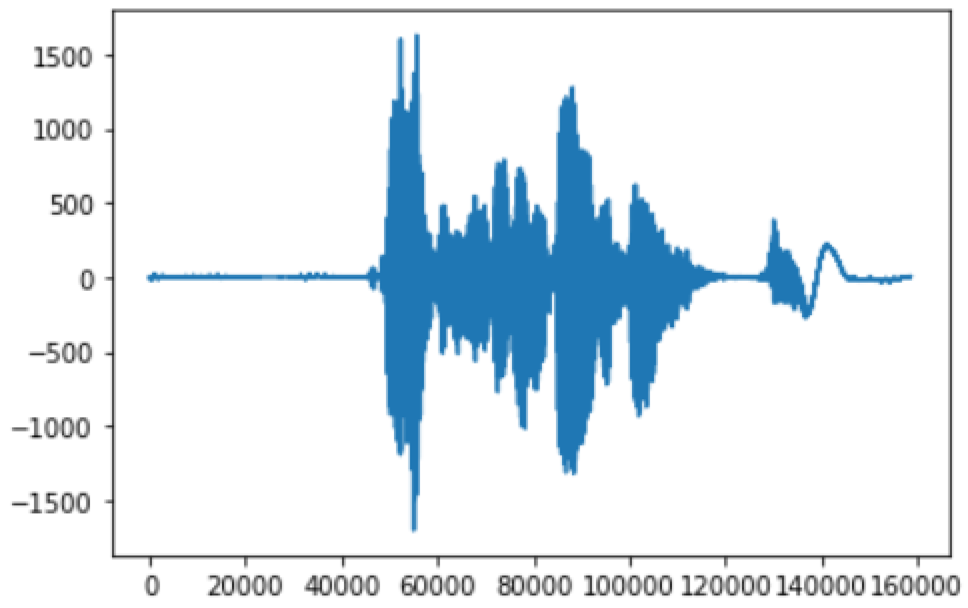


Figure 4: Audio waveform.

- **MFCC/Mel-Spectrogram were too information-dense to process visually**, but there have been attempts in literature to tune Gabor Filters to recognize shapes that correspond to emotions on spectrograms.

A Note on Feature/Data Relationship

- Not every feature works for every model, and not every model works for every feature. Need to learn what works best with what!

Data Management

- **Wanted to maximize the amount of data we had**, due to the rather small size of this dataset.
    - 90/10 Train/Validation split
    - No dedicated testbench was set out, due to the availability of regular Kaggle submissions as well as the 10% validation data used in training.

- **Our pipeline**: Load audio → Compute augmentations (shifted, noisy, original) → Feature extraction, arrangement into data matrices → train/val split into dataloaders
    - We chose these data augmentations for their ability to create new copies of data, induce robustness to different levels of noise/growliness, and enforce shift-invariance in our model predictions.
    - We also converted all stereo audio clips into mono during the augmentation step.

- **A few issues with not having a dedicated testbench**:
    - Train/val split put augmented copies of the same clip on both sides of the split → data leakage, which resulted in overly optimistic validation accuracies.
    - Difficult to reliably tune hyperparameters
    - These issues combined resulted in models that were potentially overfit to the training data.
- **How we dealt with this**
    - An advantage to data leakage: we basically had the full training set to train with, provided we reshuffle the train/val split once in a while to prevent overfitting and to let the model see all augmented copies of the data.
    - Regular submissions to Kaggle allowed us to keep track of model performance: This combined with model convergence and “val” acc of models allowed us to develop a mental heuristic for actual model performance to help adapt to this issue.
    - Using evaluation metrics in Excel such as correlation of predictions to outputs whose public Kaggle accuracies were known, as well as ensemble evaluation metrics (easy vote overlap rate, undecided rate, etc.) also in Excel.
    - Background literature on hyperparameter tuning informed us of good practices to reduce our need for tuning.

DESIGN IDEOLOGY

Because we had 4 submissions a day, we wanted to have a pipeline that could create + fit models quickly, to not waste potential submissions on the actual test dataset. This led to experimentation with features via models that were faster to train, such as SVM and MLPs.

An efficient and easy way to make sure of these regular submissions with different models was the Excel Spreadsheet Ensemble, where we took a prioritized majority vote (in a tie, priority model breaks it) among models. This was easy to implement, didn't require excessive coding, saving a significant amount of time, and progressively evolved with the number of models we made.

Idea behind ensembles was that: given the constraints, the amount of performance we can achieve from a single model is limited. Thus, we leverage the different strong suits of each model and combine them.

There were also a limited number of model types we could choose from, so a diversity in features was a priority. During the time we explored more diverse models and features, we were "burning" our daily submissions on these ensembles, trying to figure out how to get models that answered orthogonally to work together in preparation for the larger models that would come.

We will go over metrics to evaluate the ensemble as well as other models later.

METHODS

Proposed methods/models/features

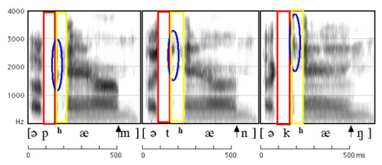
State of the art/Popular Solutions

- Currently, SotA involves using pre-trained **ResNets/CNNs on different variations of MFCC, and also TIMNet**, which is a temporal-aware bidirectional multiscale convolutional network that operates on a time-reversed copy of the input data as well - 82-85 & 92% accuracy respectively.
- Most tutorials on this dataset used a **CNN or MLP on the mean of MFCCs and achieved ~60-65% accuracy**. Some others use other features that came with librosa, the audio spectral analysis library.
- **MLPs and SVMs** were a popular choice **among us**[1] researchers, as well as the dated Gaussian Mixture Model and Hidden Markov Models.

[1] ඞ

Options available to us that we didn't take

- **Hierarchical splitting by gender, semantic content, and lower-level emotion categories**: we believe this could actually work had we more time and more data, but the need to train every model and feature 3-7 times just wasn't appealing at all.
  Here we considered employing a simple voicing detection scheme to classify the first consonant of "dogs" and "kids." However, this method was deemed to be too time-consuming than it was worth, alongside the reasons above.



Red	the stop gap in the medial phase of the /p//t//k/ (silence period)
Blue	the release burst of the /p//t//k/
Yellow	the aspiration (delay of the onset of voicing for /æ/)

Voice bar refers to a dark bar that is shown at the low frequencies and it's usually below 250Hz.Please see the voice bar of /b//d/and/g/ in intervocalic in Figure 3.3 for the voice bar of /b/,/d/and/g/.

Figure 3.3 Spectrogram of the words "a bab, a dad, a gag" for the voicing bar

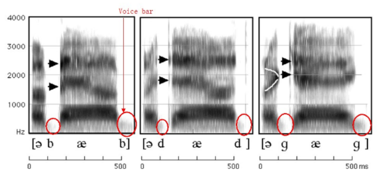


Figure 5: Voiced consonant detection using spectrogram from corpus.eduhk.hk

- **Large models**: we had no computational power to comfortably (or uncomfortably) train anything larger than about 4-5 convolution layers: our game plan was instead to "miniaturize" these large models to retain their positive characteristics.
- **Tutorial idea of using mean of MFCCs**: Using the mean of the MFCCs only allows us to capture intensity over frequency bands or intensity over time, not both. The primary issue with this feature is that it does not distinguish well between same-intensity emotions.
- **Random forest classifier**: Speech data can be shifted in time, but RF is not shift-invariant (RF wants each dimension to be a feature, thus working better with static features).
- **Training on quadratic pairs for similarity (Siamese Network)**: Idea behind this was to train the model to detect same/different emotions such that the embeddings space formed a nice 8 clusters on which an unsupervised technique could be run to achieve classification.
  Not a bad idea, but didn't align with our game plan of trying to train models up quickly and explore novel features.

A Note on Overfitting

As mentioned above, we have a data leakage problem. Testing on Kaggle helped a lot, and data augmentations, alongside shuffling our train/val split around every time we warm start, seemed to help our larger models generalize much better. Here are some things we noticed as strategies against overfitting for different kinds of models:

- **SVMs**: these guys **don't need too much training data**, and their decision boundary is pretty simple, so they **can fit less to the data and more to augmentation noise** with a simple yet nonlinear kernel. (don't use noise augmentation with SVMs)

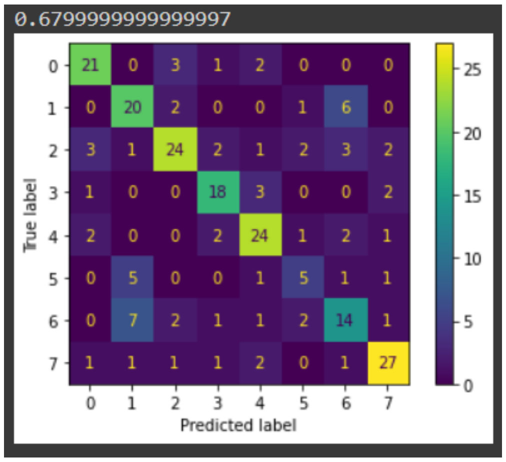
- **MLPs**: These were generally mid-level models that didn't overfit too much and didn't do too great a job. **Noising and shifting augmentations helped maintain a smaller generalization gap** from train acc to "val" and test.

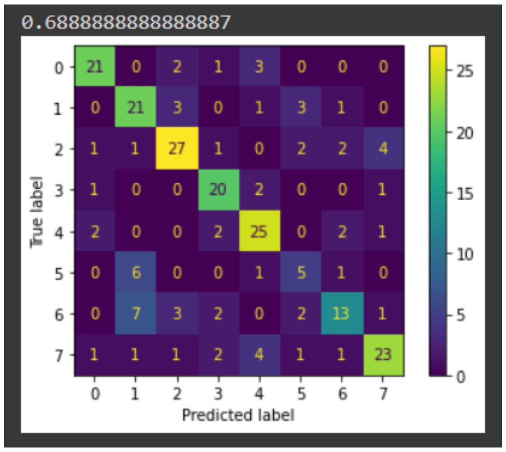
- **Large CNNs**: These took rather long to train, so attention to overfitting and convergence were especially important. We opted for manual learning rate control and SGD over Adam optimizer for these, as SGD with default parameters converges at more generalizable minima.
    - We followed the "1cycle super-convergence" and "don't decay lr, increase batch size" papers for guidelines for generalization with SGD:
        - Manual lr scheduling for convergence: (we didn't find time to write a custom scheduler)
            - Start on a moderate lr to bump out of smaller minima and start finding a "hole"
            - Once a steady loss decrease is found, increase the learning rate to quickly descend the hole. In more messy loss landscapes, this can also help avoid potholes along the way
            - Then as we reach the bottom of the hole, decrease the learning rate to settle into the minimum.
        - Learning rate and batch size for generalizability:
            - Smaller batch sizes are noisier and act as a regularizer due to this as well as the sheer number of parameter updates (cite ghost batch paper)
            - Increasing batch size and decreasing learning rate are inversely proportional and have the same effect on convergence. Can use large batches for speedy convergence, and smaller batches for generalizability.

- Techniques to combat overfitting that we didn't get to try:
    - Not having data leakage, L1/L2 regularization, dropout (mainly due to forgetting to implement this on CNNs where it actually matters - we weren't too concerned about MLPs overfitting since we only used them as ways to validate features).

BASIC PROTOTYPING MODELS: MLP & SVM

- They're fast
- It's easy to see if they converge
- Baseline accuracy of 50-60%
- **MLPs** work better for longer data (moderate-dimensional data and above), and for multimodal (concatenated) data.
    - MLPs have a lot of params and are all-seeing: can combine features effectively and don't have tight bottlenecks/blindspots
- **SVMs** are useful in their simplicity. They are hard to overfit (except in the Noisy SVM Case). They work better for lower-dimensional (e.x. 1 time-series feature), and have different decision boundaries than MLPs, making them diverse and thus good (mix of SVM and MLPs) when used together in an Ensemble.
- **Polynomial** with **degree 2** works best for the data. Inflated "val" accuracy of **65-77%**, which translates to **54-64%** on real Kaggle test accuracy. (Degree 2, according to Wikipedia, is also the most common kernel for speech applications)
- We tested **RBF**, **linear**, **degree 3-5**, they all were worse. Linear came close second on select features. After experimenting with many gamma values, we chose 1 and the default "scale" option, which was 1/n_samples * variance in data. We got the best performance from a C of 1.





Figures 5 and 6: Accuracy and confusion matrix for **Binomial SVM** (left) and **Linear SVM** (right) on 4-split db-4 Wavelet MFCC + MFCC, PCA'd to 300 dimensions (will be covered in later sections). While the Linear SVM can sometimes have a higher "validation" score, its Kaggle score is always lower than Binomial. This is a result of overfitting to the augmentation noise, as well as the tendency for a simpler decision boundary to underfit the data, not having enough complexity to explain the commonalities between seen and unseen data.

For the basic MLP structure, we used an MLP of 5 layers, with the following neuron configuration: **input → 2048 → 2048 → 1024 → 256 → 8.**

Basic Features: Mel-Spectrogram & MFCC

Mel-Spectrogram - giant pretty pictures: Basically log-scale STFT passed through Mel filters.

- Computed using the trusty and fast STFT, also can be unrolled or used as an image.
- Problem: dimensionality was too large for MLPs to reasonably converge! (possibly too large to comfortably load into GPU RAM as well)

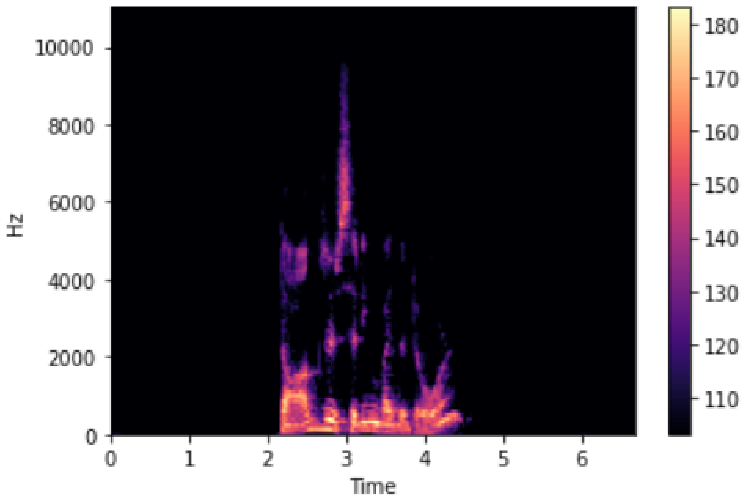


Figure 7: Mel-Spectrogram of Speech Audio.

MFCC - the poster child of Speech Emotion Recognition: DCT of log-Mel-Spectrogram

Intuitive explanations for why MFCCs (Mel-freq cepstral coefficients) are a good solo feature:

- They can **efficiently capture most data inherent to a spectrogram in a lower dimensionality** while also being able to **mimic human perception with the log-mel scale.** (Packs similar information as spectrograms into 20 coefficient arrays via the DCT)

- They can be unrolled or used as an image, dimensionality was not too large and not too small, it was juuuust right! - This feature worked with basically everything, and became the basis for our other features, as well as a lot of features in recent speech recognition literature.

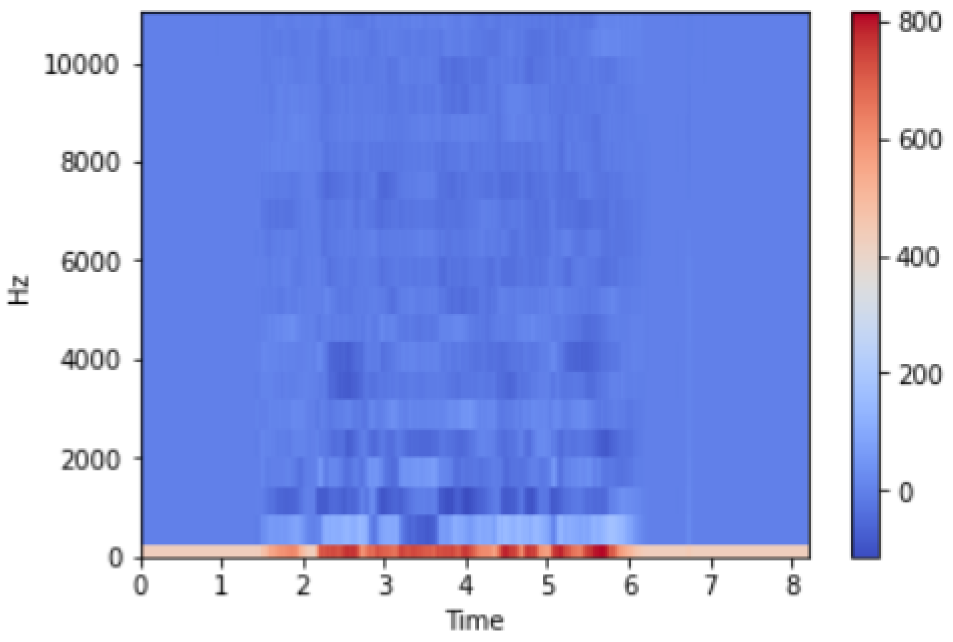


Figure 8: MFCC (Mel-Frequency Cepstral Coefficient) of Speech Audio.

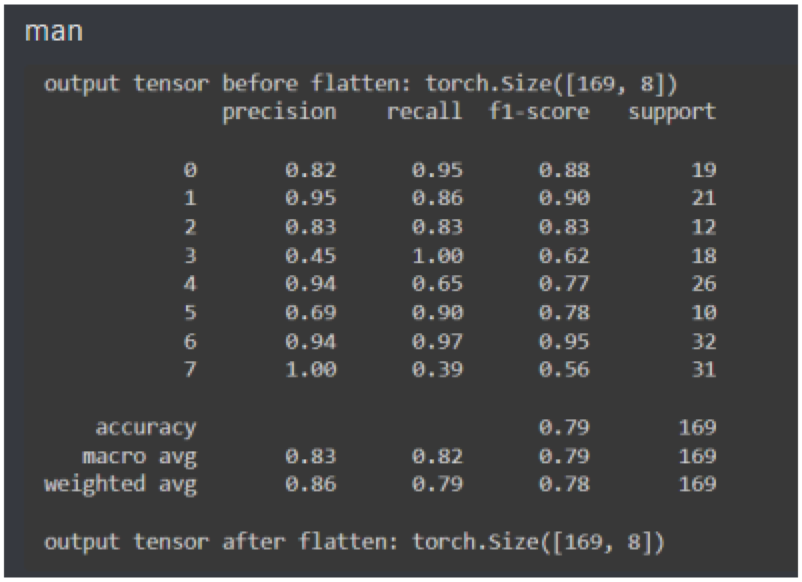
We could use other forms of spectra (STFT, Wavelet scalogram, Hilbert spectra) as input but ultimately decided to just MFCC-ize everything and use MFCC as a standard to compare to, as the feature shape/dimensions of MFCCs were similar to each other, making it easier to optimize models for. For MLPs, we started out with the first 13 MFCCs in concern of dimensionality, however we later on used the full 20 MFCCs for further calculations.

With Unrolled MFCC on MLPs and SVMs, we achieved an accuracy of 55-60% on Kaggle. With this, we were able to enjoy 1st place for half a week before gaining a burst of external motivation to try something better :)

Here we tried gendering our data as well: we trained a gender detector to **76%** accuracy using average fundamental frequency (pitch), but assumed 100% classification eventually (this is the standard result in literature) and sorted our data based on hand-labeled gender just to test. Results: (on SVMs/MLPs trained on MFCC)

- Gendering improved accuracy on female speakers by **2-4%** likely due to unity in the way each emotion was expressed (literature empirically supports this viewpoint)
- Gendering decreased male accuracy by **2-4%**, likely due to lack of training data, and the ability for men to express negative-sentiment emotions in more clearly softer and harsher ways, which did not help classification.
- Overall made negligible difference when combined.

Here, even with inflated validation accuracy due to data leakage, we found that consistently, across MLPs and SVMs, **female emotions were recognized more often than male emotions**.



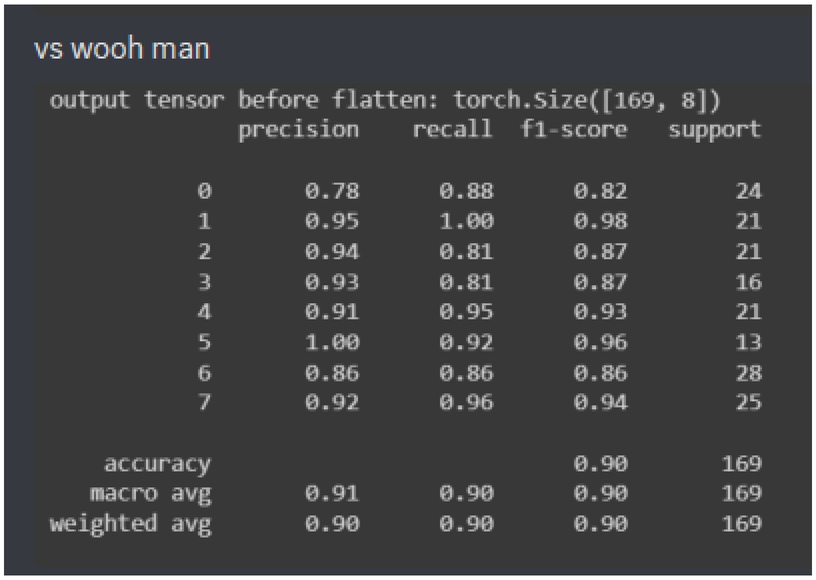
man

```
output tensor before flatten: torch.Size([169, 8])
              precision    recall  f1-score   support

           0       0.82      0.95      0.88        19
           1       0.95      0.86      0.90        21
           2       0.83      0.83      0.83        12
           3       0.45      1.00      0.62        18
           4       0.94      0.65      0.77        26
           5       0.69      0.90      0.78        10
           6       0.94      0.97      0.95        32
           7       1.00      0.39      0.56        31

    accuracy                           0.79       169
   macro avg       0.83      0.82      0.79       169
weighted avg       0.86      0.79      0.78       169

output tensor after flatten: torch.Size([169, 8])
```



vs wooh man

```
output tensor before flatten: torch.Size([169, 8])
              precision    recall  f1-score   support

           0       0.78      0.88      0.82        24
           1       0.95      1.00      0.98        21
           2       0.94      0.81      0.87        21
           3       0.93      0.81      0.87        16
           4       0.91      0.95      0.93        21
           5       1.00      0.92      0.96        13
           6       0.86      0.86      0.86        28
           7       0.92      0.96      0.94        25

    accuracy                           0.90       169
   macro avg       0.91      0.90      0.90       169
weighted avg       0.90      0.90      0.90       169
```

Figures 9 and 10: **MLP** on **MFCC** classification report for **male (top)** and **female (bottom)**. Even with inflated accuracies, one can see that even on noisy versions of the training data, the male model is struggling to fit compared to the female model.

Going Beyond the Basics

Here we wanted good ways to distinguish between same-intensity + same-sentiment emotions.

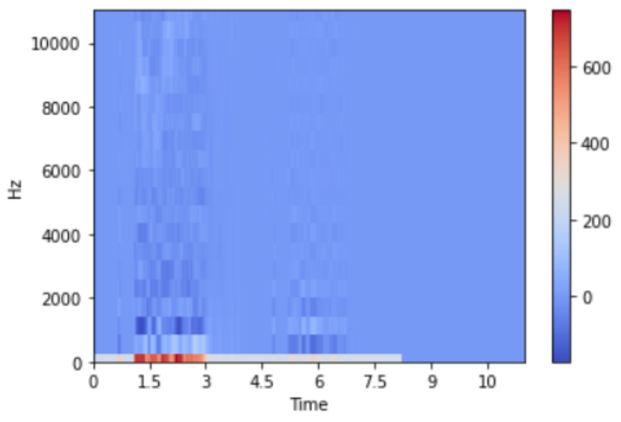
Landmark Improvement: WAVELETS

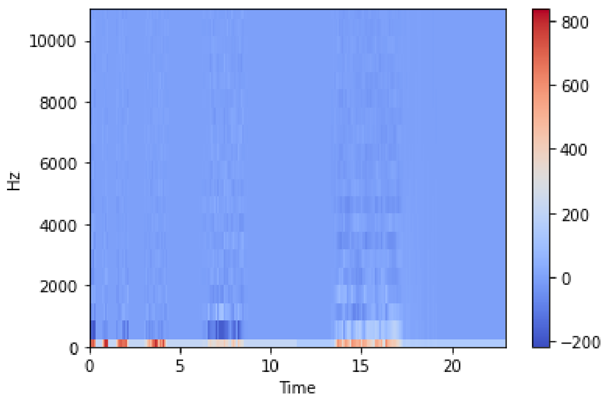
Wavelet transformation of the audio allows for the input to be split into different levels of detail, and offer higher time resolution with higher frequencies, which means they highlight small changes in the formants range better.

A Note on Padding: Since wavelets are not shift-invariant, we moved from padding the MFCC spectrum to padding the audio signals instead. This gave our wavelet-based models the ability to potentially learn to ignore the effects of shifting the same audio signal in wavelet form. We later realized that this was a better move, as it allowed for more consistent separation between features across audio clips, which came in handy for our multimodal models.

Daubechies DWT Multi-Resolution Analysis

(a) Wavelets are localized in time -- allowing for better time-series resolution than Fourier. (b) Wavelet transform can help separate data into low and high-frequency sets, and thanks to their increased time resolution at higher frequencies, are able to see the small high-frequency details that Fourier tends to miss (c) the application of MFCC on the concatenated multiscale DWT analysis is discussed to great detail in literature, providing us with a learning opportunity and a solid basis to try things upon.
SVM showed a 7% accuracy increase when performing Wavelet-MFCC **reaching 64%** on Kaggle, and **MLP was more able to consistently hit 60%**, which was the previous “peak”. Note that we primarily used a one-split DB4 wavelet. This W-MFCC feature became the defining element of our project.





Figures 11 and 12: Wavelet-based MFCC with 1 split (left) and 4 splits + MFCC (right). Lengthy duration was due to padding. In the 5-split image the halved length of subsequent splits can be clearly seen.

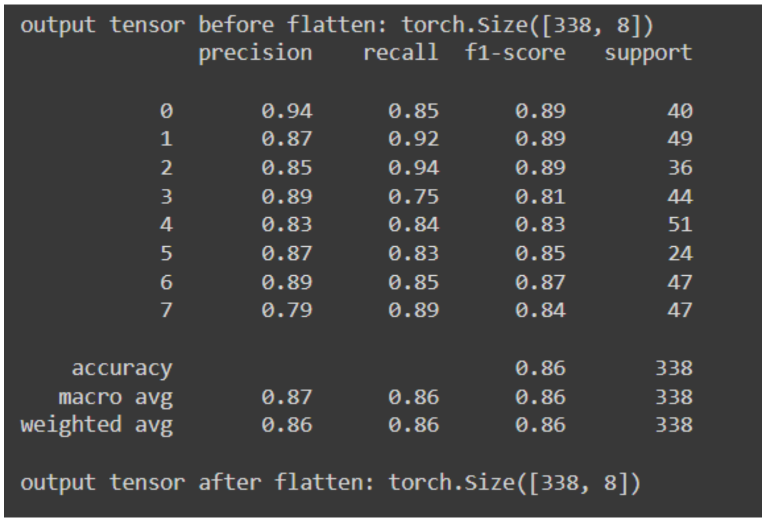
An Attempt at Complex Gabor

We also tried a continuous wavelet with Complex Gabor. The reason for this was that Gabor was closer to shift-invariance than Daubechies. We would take the magnitude of the multiscale decompositions, and use that to compute MFCCs again (since MFCCs are highly data-efficient). Glaring Issue: Have to pick center frequency, and bandwidth parameters. Specify which part of your signal you want. We found this to be a hassle to tune, so decided to stick with db4 for the sake of time.

While Complex Gabor offered similar accuracy as db4 (**~60%** accuracy on MLP and **57%** on SVM), the answers from Complex Gabor did have a low correlation to MLPs and SVMs using other wavelets, however, so there was still value in including these in some ensembles even bereft of proper tuning, which we believed to be the main cause of such a poor performance.

Trying incorrectly-computed LPCC/WLPCC to varying degrees of success

We tried to use LPC and its MFCC derivative as well. Instead of computing LPCC via the correct formula, we computed MFCC of LPC coefficients (different from LPCC). Default window length of 512 meant **we only got ONE coefficient of the MFCC on LPC coefficients**. Without knowing this, we attempted to toss it into models alongside **MFCC, wavelet MFCC, and the wavelet transform of this wrong LPCC**. This actually gave it better predictions as well as more diversity, allowing our **Multimodal MLP** (this was the point in time when we coded a multimodal MLP **consisting of 3 parallel blocks**, each doing analysis on all coefficients within a certain frequency range) to **reach 65% on Kaggle**. So, it worked? The inclusion of “LPCC” became the new feature for further training.


```
output tensor before flatten: torch.Size([338, 8])
              precision    recall  f1-score   support

           0       0.94      0.85      0.89        40
           1       0.87      0.92      0.89        49
           2       0.85      0.94      0.89        36
           3       0.89      0.75      0.81        44
           4       0.83      0.84      0.83        51
           5       0.87      0.83      0.85        24
           6       0.89      0.85      0.87        47
           7       0.79      0.89      0.84        47

    accuracy                           0.86       338
   macro avg       0.87      0.86      0.86       338
weighted avg       0.86      0.86      0.86       338

output tensor after flatten: torch.Size([338, 8])
```
Figure 13: Classification report of MLP on “val” data after the inclusion of “LPCC”

For the Multimodal MLP, we tested a configuration of 2-3 parallel blocks as follows:

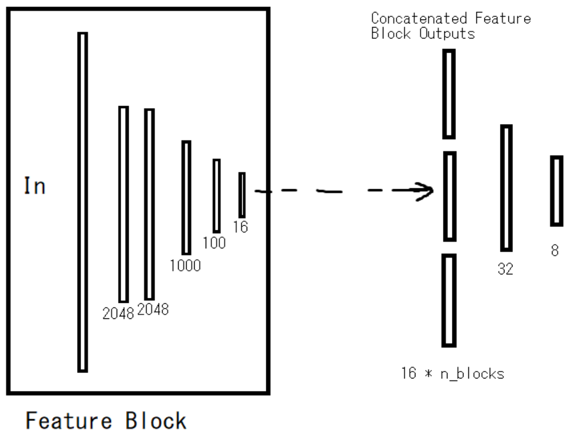


Figure 14: Multimodal MLP parallel configuration.

At this point in time, our ensemble approach with the “LPCC” MLP was also the highest accuracy we had, which gave us first place for another 2 days, before the coming of CNNs…

Also, in hindsight, we realized that concatenating different coefficient arrays of wavelet multiscale analysis then taking the MFCC does not take into account edge behavior when two numbers on different scales are being treated in the same window as though they were on the same scale… Luckily things panned out well for us; however, we would not be surprised if this caused the W-MFCCs to not perform at their full potential.

Failures

“Tried-and-True Architecture”: ResNets

We tried using the skeleton of a ResNet-50 we had from another project that had been known to work (since at this point we were not at all confident with coding our own CNNs and dealing with dimensionality errors), changed 2D convolutions to 1D and reducing the number of blocks to 3 (so more like a ResNet-14 now), and we were able to achieve **54-56% accuracy with very unoptimized training regimes**, for ungendered and gendered respectively. Then, we soon forgot about this untuned and nonoptimal model, at least for the time being…
Problem: we used an **architecture clearly not built for our purpose**, dataset, and computational limits (kept crashing Colab GPU/CPU); also the training regime was so terribly unpolished.

0-1D Low-Level Descriptors and RFE (Recursive Feature Elimination)

The idea behind this was to form a useful ensemble with high diversity from many low-level features. It did not pan out. But we learned some stuff: we learned that because the MFCCs have 20 features packed into 1 feature, that it was a lot better than these.

In accordance with our linguistics research, we believed that jitter and shimmer would be good metrics (since they are responsible for noisiness/timbre), but they were only as good as blind guessing. We believe this is from how varied peoples' voice characteristics can be; useful if one knows the average jitter and shimmer of a person, but those are unknown for our observations. Perhaps these features would've been good for the gender detector we never finished (literature says timbre and pitch are the 2 defining characteristics).

RFE Greedy Algorithm:
We loaded up low-level descriptors from OpenSmile (25 time-series features, 523 dimensions each). We ran a greedy algorithm to:

- select the highest accuracy feature via **SVM** and **RF**
- eliminate other features whose correlation "scores" (correlation of predictions to the selected feature, penalized by low accuracy) were higher than a predetermined threshold
- select the highest accuracy feature among those remaining, repeat.

The accuracy-adjusted correlation formula is shown here:

```
corr = pointwise_similarity(picked_predictions, candidate_predictions)
adjusted_corr = corr/ candidate_accuracy
```

The idea behind this scaling was that a higher correlation should be tolerated if the candidate (one you're trying to decide whether to eliminate) has higher accuracy, as there is a higher chance that there is more overlap, simply due to both prediction sets being more correct.

After running, the greedy algorithm eliminated 16, leaving 9, out of which we formed random subsets of 2, 3, 4, 5, then picked the **highest RF/SVM accuracy** among these subsets, their db-4 wavelet transforms and their PCA's to 100, 200, 315 dimensions, and the highest accuracy to ever be achieved was **36%**. Convergence testing on MLP affirmed this.

We also tried doing the same RFE on an 88-dimensional vector consisting of mean/stdev as well as other 0-D features of the above 1D features, also loaded from OpenSmile, and achieved similar results.

We also evaluated **fundamental frequency** (spectral rolloff at 0.01 - the point in frequency below which 1% of the energy was concentrated), its **wavelet transform**, as well as **MFCC mean/stdevs,** but to no avail.

Notes: we forgot to try derivatives of MFCC, those could be good. They showed promise during the gender detector trial, and also the 1D feature set did have MFCCs 1-4 but still were not nearly as good as all 20 together.

Feature selection using PCA

Our issue with PCA was that it is blind. While PCA can help you find the orthogonal and most high-variance-explaining directions in your data, what if these “strongest” directions are instead irrelevant, and the finer details in the dimensions with smaller singular values are actually what you want? That would be bad. But we tried anyway, and even while concatenating two good features together, we were better off with just not using PCA.

Overcoming the Fear Of Dimensionality Errors - The Advent of CNNs

After experimenting with dimensionality on the **ResNet** and realizing that errors only made us stronger, we read up on the PyTorch documentation and made our own **CNNs**. From this point onwards, we truly had ascended and further matured as ML researchers, and thus were taking bolder strides towards achieving victory. We had no fear left in us and were constrained only by 1) Jeff's deadlines and 2) Colab's computational limits.

Modular Design with Blocks and Multimodal CNNs

Idea was to put all convolutional computation into blocks and combine the blocks later in the main model class through fully-connected layers. The idea was to make it more flexible if we wanted to keep different kinds of blocks for different features, and was especially useful since we were dealing with multimodal data and multiscale analysis.

(We in hindsight thought there was a problem of the multiscale analysis not being on the same frequency scale yet were being concatenated together horizontally, however we trusted that with enough convolution filters the CNN may see the concatenated features as different parts of an image)

At first, because we didn't know how to manage memory properly, we kept appending to lists in the GPU, which made it crash. Later replaced with List Comprehension (effective append-less way of making lists)--fixing it made a big difference.

1D Multimodal CNN

We concatenated the **MFCC**, **W-MFCC**, and “**LPCC**” together and unrolled them onto a **1D CNN** with 3 parallel blocks, intending to split the data vertically to roughly isolate each feature. However, since we unrolled our data row-wise, the split actually amounted to splitting along the frequency axis instead, which ended up working out well for us, as we achieved a Kaggle accuracy of **71%.**

We followed the 1cycle paper and trained with a batch size of 64 on 70 epochs of lr=1 and 20 more epochs of lr=0.3 for top accuracy (this allowed our models to converge rapidly, and at the time was our best way of achieving generalizability). This style of model was inspired also by the “1D CNN for time-series predictions - a baseline” paper. This improvement once again improved our ensemble performance.

2D Semi-Global Residual Attention Network (Lovingly called WiggleNet) (or WiggLeNet)

The name comes from the idea of convolution filters being filters on a camera lens, and the large semi-global kernel acting like the cameraman moving (wiggling) the camera around to capture the whole scene chunks at a time. Another advantage this model had over our previous models was our choice to do a 2D convolution, which meant that relationships of neighbors in the vertical direction were no longer ignored.

With this model we implemented a large receptive field and utilized global average pooling to learn attention mappings as well as pick out general shapes in the image, followed by differently-sized convolutions for multiscale analysis. We implemented downsampled residual connections to keep gradient flow at a high level. This model was a very fun exercise of miniaturizing much larger architectures to keep their defining functionality while also fitting inside our limited GPU RAM. Originally, we wanted to have a global receptive field and 1x1 downsampling convolutions, but we found that such a downsampling scheme was too harsh to retain enough information to converge, and so we switched to the following architecture:

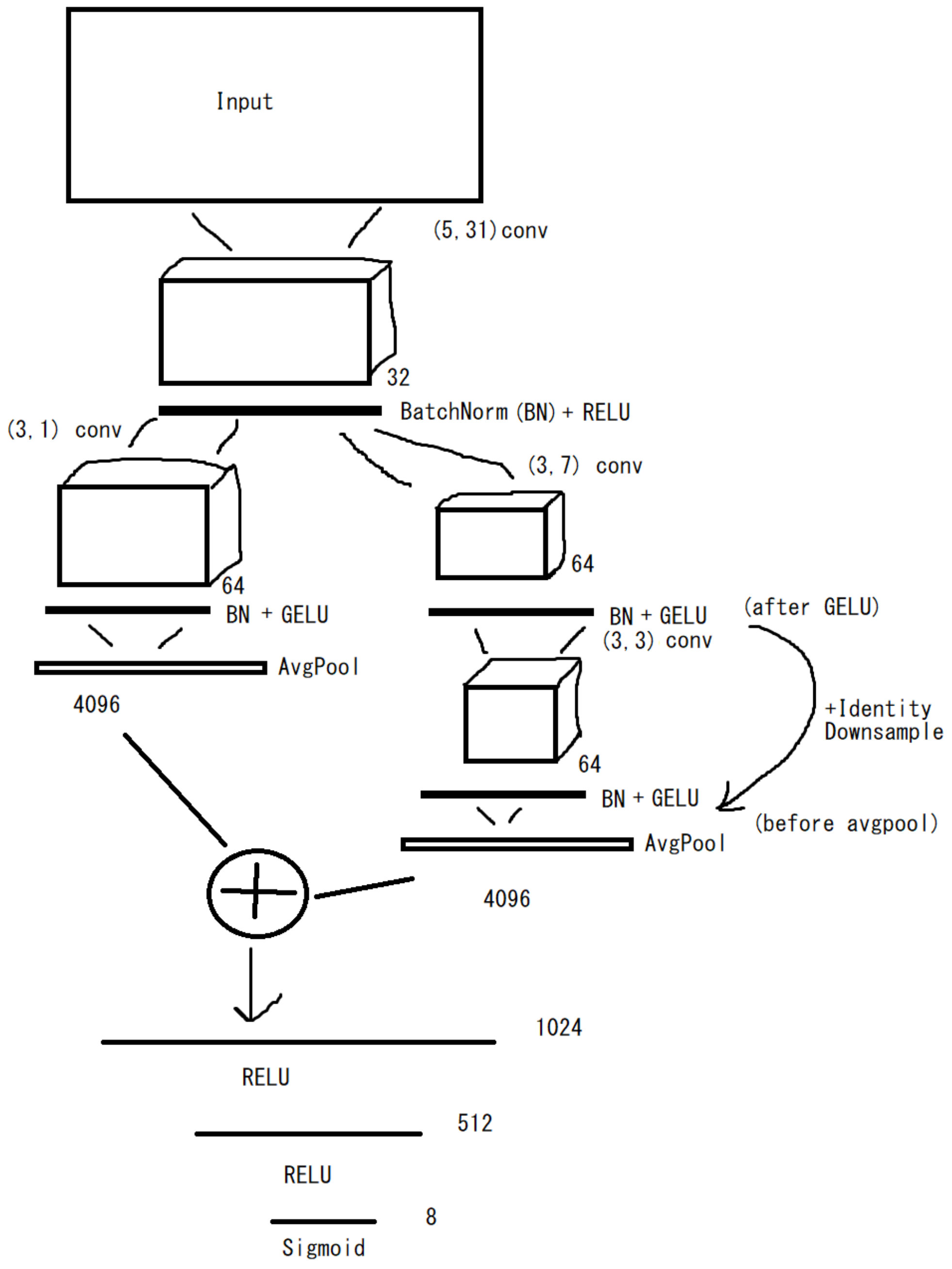
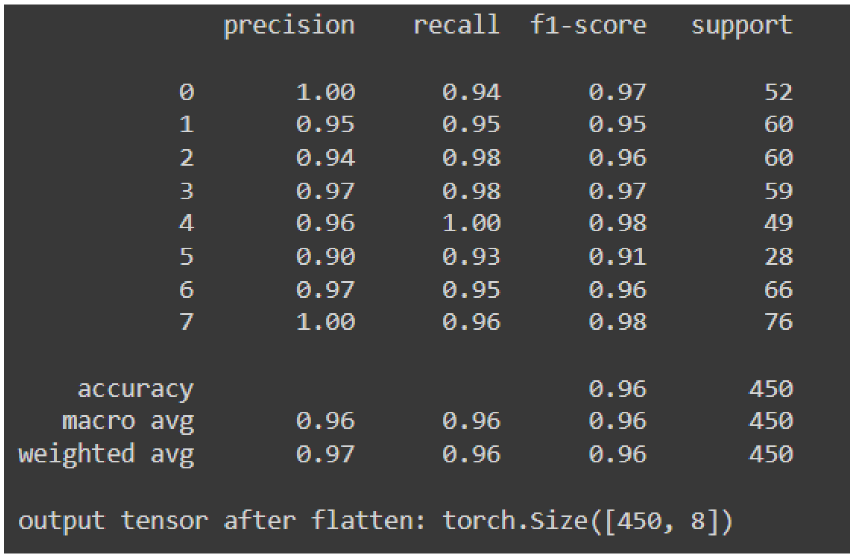


Figure 15: Intended Multiscale Residual Attention Network Architecture.

Unfortunately, due to typos in the code, the AvgPool layer after the (3, 1) convolution layer was instead applied after the (3, 7) convolution layer, and the output of the (3, 1) layer was not a factor in the input. We will re-run the code to see if our model gets better. However, the typo'd model still proved to work quite well.

Removing the “LPCC” and **testing only on MFCC + W-MFCC** (after realizing what “LPCC” we were using) as our most data-packed features, **we trained for only 19 epochs on lr=0.001 on batch size=4 and achieved a Kaggle accuracy of 79.8%**. At this point, while putting this model into our ensemble yielded different results that would affect the private Kaggle accuracy, public accuracy remained unchanged. One difference in the W-MFCC we calculated was that we used a 2-split Db-4 instead of a 1-split, hoping that it would offer more detail. Perhaps this model’s success can partly be attributed to this as well.



```
              precision    recall  f1-score   support

           0       1.00      0.94      0.97        52
           1       0.95      0.95      0.95        60
           2       0.94      0.98      0.96        60
           3       0.97      0.98      0.97        59
           4       0.96      1.00      0.98        49
           5       0.90      0.93      0.91        28
           6       0.97      0.95      0.96        66
           7       1.00      0.96      0.98        76

    accuracy                           0.96       450
   macro avg       0.96      0.96      0.96       450
weighted avg       0.97      0.96      0.96       450

output tensor after flatten: torch.Size([450, 8])
```

Figure 16: Classification report on “val” data for WiggleNet on WMFCC + MFCC

Hilbert-Huang Transform: A Better (ish?) and Data-Driven Time-Frequency Representation via Empirical Mode Decomposition

Upon searching for better features to use, we came across EMD and the HHT. The idea is to achieve a time-freq representation that offered a more flexible choice of resolution in time, just like the advantage Wavelets had over Fourier, but also a time-freq representation whose basis functions we could learn from the data and not have to choose a priori.

With EMD (using the python “emd” package), we were able to iteratively decompose our audio signals into Intrinsic Mode Functions (IMFs) which would serve as basis functions on which the Power Spectral Density of the Hilbert Transform could be computed, and then summed up to achieve a 2D representation. The procedure and further explanation of Hilbert-Huang can be found here:

We can apply the “sifting” algorithm to break a signal into its IMFs.

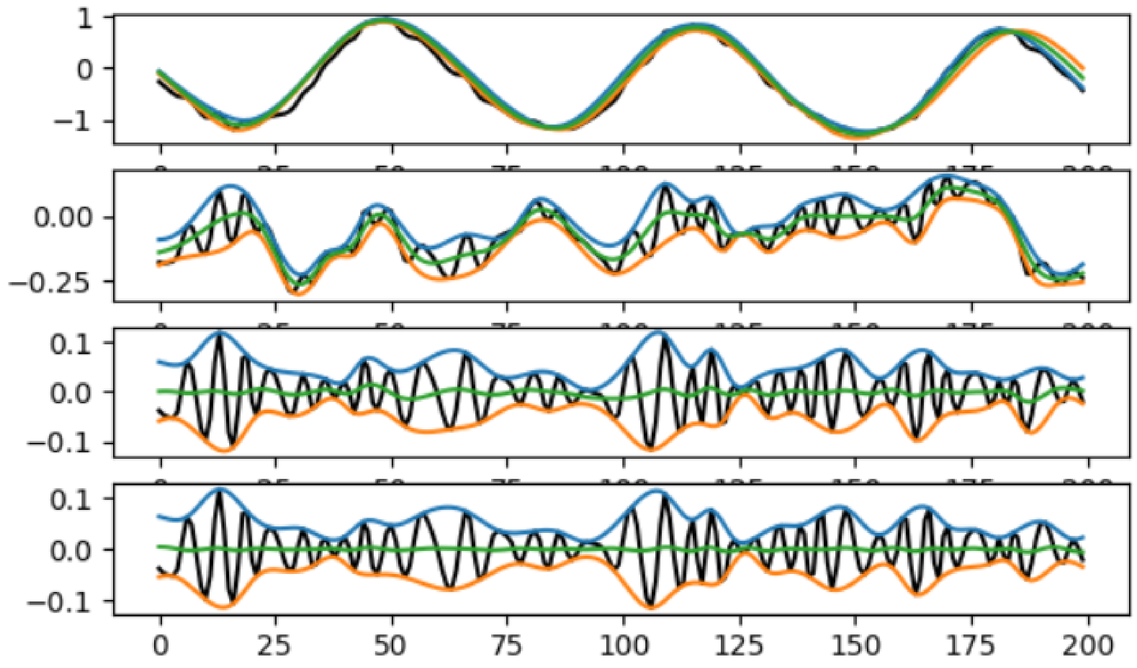


Figure 17: Sifting Process on A Noisy Sine Wave

First we calculate the mean between the top and bottom envelopes of the signal and subtract that from the signal. This mean will be the **first IMF**, and what's left after the subtraction we repeat this procedure on until the remaining signal's mean is 0. Each time we do this we get an IMF. Since the expected runtime is data-dependent, most programs allow the user to set a cap for how many IMFs they want. We chose 7 for this application, as it is a number found to be suitable for speech in literature while also not taking so long to compute.

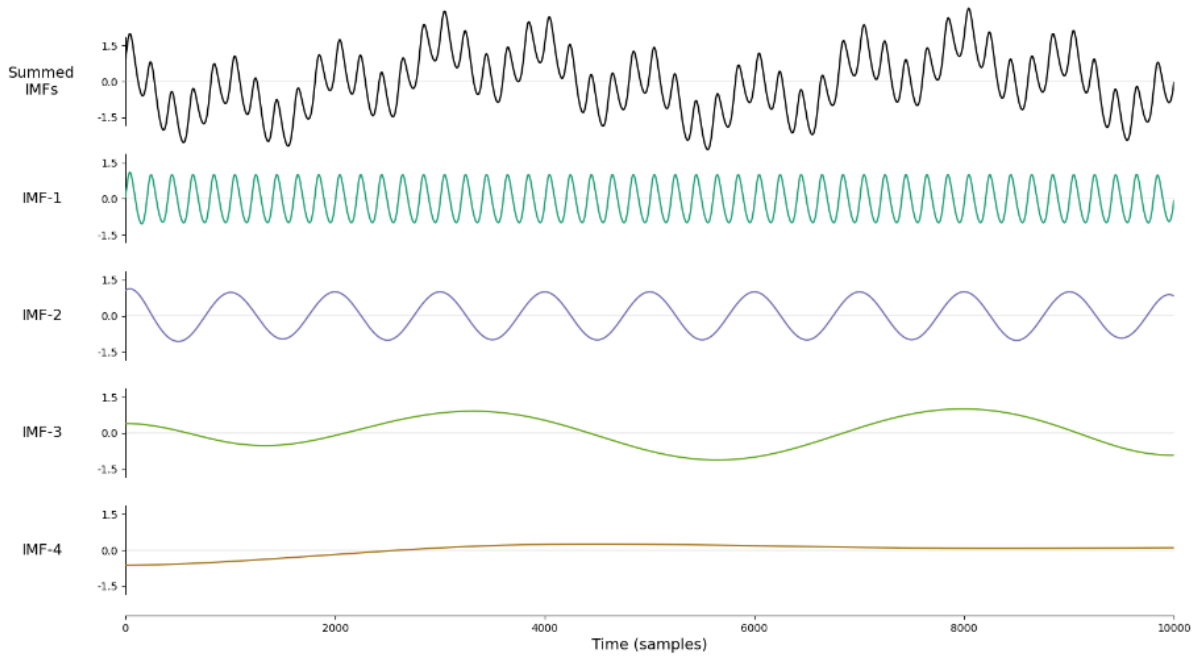


Figure 18: **IMFs** resulting from applying **EMD** on a sum of sine waves. This method is very time consuming, so it's rather inefficient for LTI systems when something faster exists that is just as good. However, for non-linear non-stationary signals, this is a viable method for time/freq resolution.

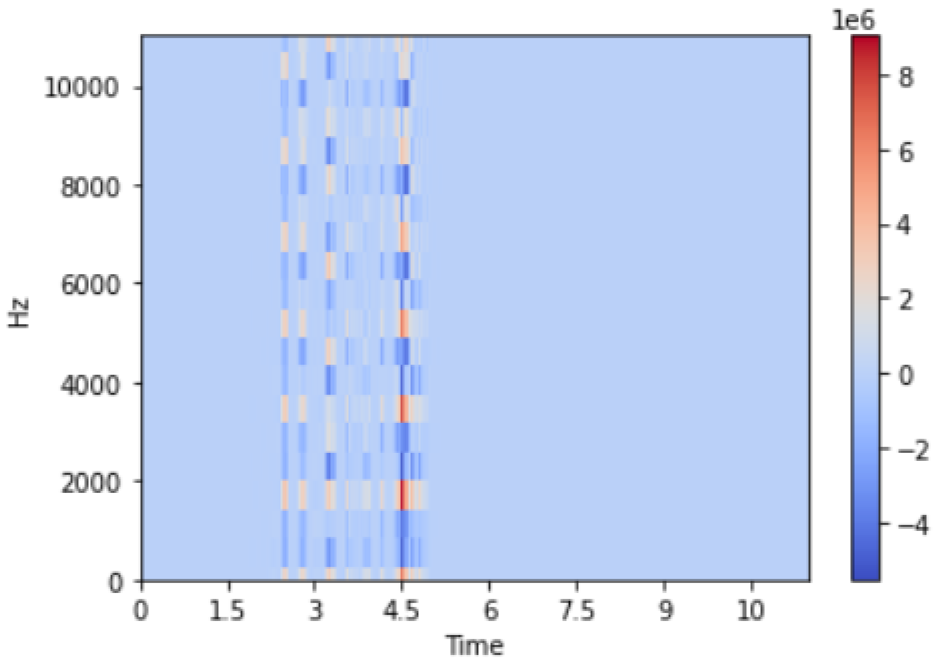
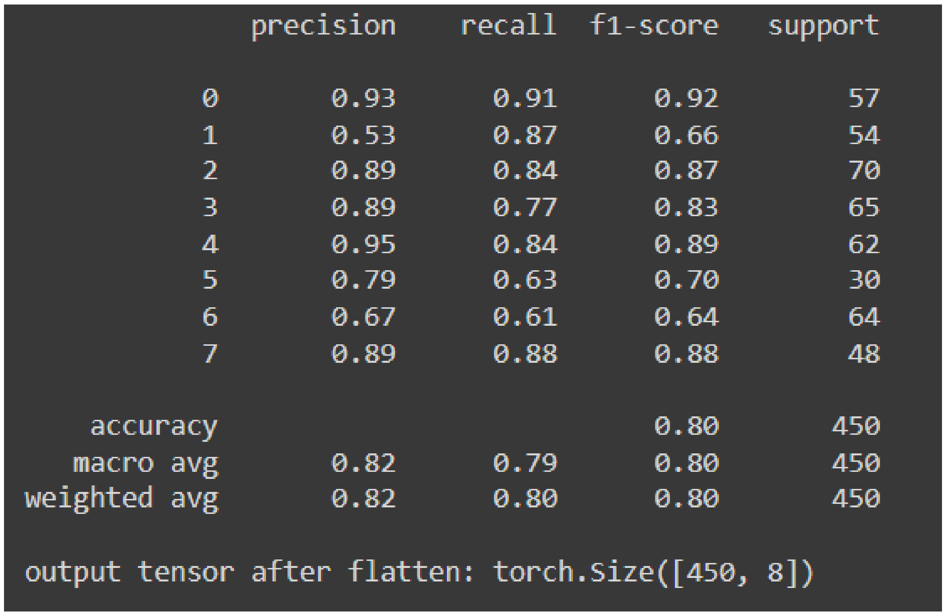


Figure 19: MFCC representation of Hilbert Spectrum. Taken by downsampling → log scale → DCT.

Problems: Since the Hilbert-Huang Transform has the same length as the audio signal, we had to downsample the spectra by a factor of 512 (to keep the same size as MFCCs for comparison, but also because of GPU memory and running time issues). Also, most sources we had were academic papers, meaning that this feature is less widely-used, a result of EMD being an entirely time-domain and iterative process, meaning that it has an excruciatingly long runtime. Feature extraction with 70 frequency histogram bins (on 4500 training data samples), downsampling, and converting to MFCC took a total of over 10 seconds per sample, meaning that **feature extraction alone, on the coarsest possible resolution, took over 8 hours!**

Because of this coarse resolution (we didn't have the time to go any finer), our model accuracy, trained on the same Attention Network, only reached 50%. We believe that with more resolution/tuning as well as more powerful HHT-MFCC features, such as combining with instantaneous-frequency-weighting and the Teager Energy Operator, that we would get results as competitive as MFCC + W-MFCC with Attention Network.

We also believe that with more careful model/hyperparameter tuning/selection, that the Attention Network would outperform its small-kernel CNN competition.



```
              precision    recall  f1-score   support

           0       0.93      0.91      0.92        57
           1       0.53      0.87      0.66        54
           2       0.89      0.84      0.87        70
           3       0.89      0.77      0.83        65
           4       0.95      0.84      0.89        62
           5       0.79      0.63      0.70        30
           6       0.67      0.61      0.64        64
           7       0.89      0.88      0.88        48

    accuracy                           0.80       450
   macro avg       0.82      0.79      0.80       450
weighted avg       0.82      0.80      0.80       450

output tensor after flatten: torch.Size([450, 8])
```

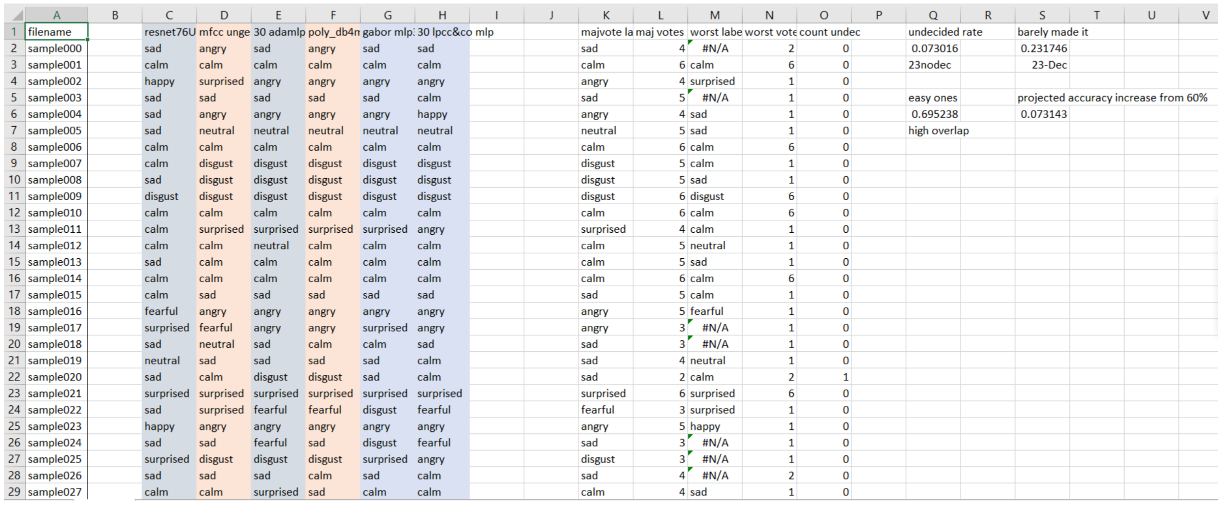
Figure 20: Classification report on "val" data for WiggleNet on Hilbert MFCC.

Ensemble Learning

We used an Excel spreadsheet to document the predictions of every model. This was a good practice, as we had access to the predictions to at least use for majority voting even if we lose the model file. We conducted simple majority voting with the tiebreaker model selected by simply putting it in the tiebreaker slot(s). We evaluate our ensembles on the following metrics:

- Undecided rate: the ratio of ties over total
- Easy answer overlap: the ratio of majority answers with **> 50%** of the votes
- "Barely made it" rate: the ratio of votes that were decided, however have the possibility to be swayed. Essentially 1 minus the sum of the above two.

We noticed that there were patterns as to which sample clips were easier to answer and those that were more difficult to answer by paying attention to the tabulated results. A model tends to get better as its "easy answer overlap" (confidence) score increases, but it also comes with the reluctance to change its answer, limiting its potential to further evolve.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
1	filename		resnet76U	mfcc unge	30 adamlp	poly_db4n	gabor mlp:	30 lpcc&co mlp			majvote la	maj votes	worst labe	worst vote	count undec		undecided rate		barely made it			
2	sample000		sad	angry	sad	angry	sad	sad			sad	4	#N/A	2	0		0.073016		0.231746			
3	sample001		calm	calm	calm	calm	calm	calm			calm	6	calm	6	0		23nodec		23-Dec			
4	sample002		happy	surprised	angry	angry	angry	angry			angry	4	surprised	1	0							
5	sample003		sad	sad	sad	sad	sad	calm			sad	5	#N/A	1	0		easy ones		projected accuracy increase from 60%			
6	sample004		sad	angry	angry	angry	angry	happy			angry	4	sad	1	0		0.695238		0.073143			
7	sample005		sad	neutral	neutral	neutral	neutral	neutral			neutral	5	sad	1	0		high overlap					
8	sample006		calm	calm	calm	calm	calm	calm			calm	6	calm	6	0							
9	sample007		calm	disgust	disgust	disgust	disgust	disgust			disgust	5	calm	1	0							
10	sample008		sad	disgust	disgust	disgust	disgust	disgust			disgust	5	sad	1	0							
11	sample009		disgust	disgust	disgust	disgust	disgust	disgust			disgust	6	disgust	6	0							
12	sample010		calm	calm	calm	calm	calm	calm			calm	6	calm	6	0							
13	sample011		calm	surprised	surprised	surprised	surprised	angry			surprised	4	calm	1	0							
14	sample012		calm	calm	neutral	calm	calm	calm			calm	5	neutral	1	0							
15	sample013		sad	calm	calm	calm	calm	calm			calm	5	sad	1	0							
16	sample014		calm	calm	calm	calm	calm	calm			calm	6	calm	6	0							
17	sample015		calm	sad	sad	sad	sad	sad			sad	5	calm	1	0							
18	sample016		fearful	angry	angry	angry	angry	angry			angry	5	fearful	1	0							
19	sample017		surprised	fearful	angry	angry	surprised	angry			angry	3	#N/A	1	0							
20	sample018		sad	neutral	sad	calm	calm	sad			sad	3	#N/A	1	0							
21	sample019		neutral	sad	sad	sad	sad	calm			sad	4	neutral	1	0							
22	sample020		sad	calm	disgust	disgust	sad	calm			sad	2	calm	2	1							
23	sample021		surprised	surprised	surprised	surprised	surprised	surprised			surprised	6	surprised	6	0							
24	sample022		sad	surprised	fearful	fearful	disgust	fearful			fearful	3	surprised	1	0							
25	sample023		happy	angry	angry	angry	angry	angry			angry	5	happy	1	0							
26	sample024		sad	sad	fearful	sad	disgust	fearful			sad	3	#N/A	1	0							
27	sample025		surprised	disgust	disgust	disgust	surprised	angry			disgust	3	#N/A	1	0							
28	sample026		sad	sad	sad	calm	sad	calm			sad	4	#N/A	2	0							
29	sample027		calm	calm	surprised	sad	calm	calm			calm	4	sad	1	0							

Figure 21: Example of Excel Sheet Ensemble. Metrics in top right corner (proj.inc. is experimental).

During our testing we evaluated a few key ensembles:
(Here we will refer to WMFCC + MFCC as W+MFCC and Multimodal CNN + WiggleNet as custom-made CNNs)

- 2x 1D ResNet (76 & 84 epochs), MFCC Binomial SVM, 2x MFCC MLP (gendered & ungendered): **58%** accuracy

- 1D ResNet (76 ep), MFCC MLP, Db-4 MFCC MLP, Db-4 Binomial SVM, Gabor MLP, LPCC + W+MFCC MLP: **68%** accuracy

- Previous ensemble as 1 model, LPCC + W+MFCC Multimodal MLP, Db-4 Binomial SVM, Gabor MLP, LPCC + W+MFCC MLP: **69%** accuracy

- The previous ensemble as 1 model, LPCC + W+MFCC Multimodal MLP, Db-4 Binomial SVM, Gabor MLP, LPCC + W+MFCC MLP, MFCC Mean/Stdev Binomial SVM: **70-72%** accuracy: this was our **highest accuracy without using our custom-made CNNs**.

- All models in the previous ensemble, added on top our 1D Multimodal CNN (tiebreaker) on LPCC + W+MFCC: **74.4%** accuracy, which was a **3% increase** from the CNN of focus. Further attempts at making an ensemble did not change the accuracy by a significant amount. We theorized that this “gene pool” needed better specimens to keep evolving.

- CNN Club - Ensemble between all the CNNs including 2x W-MFCC WiggleNet (tiebreaker), 1D ResNet, 2x 1D Multimodal (90 & 155 epochs): **79.8%** accuracy - no noticeable difference to W-MFCC WiggleNet on the public dataset.

- We also found that putting every model together for a majority vote (excluding WiggleNet) yielded an impressive **73%** accuracy, showcasing that while democracy can be dragged down by bad decision-makers, a democratic society for the most part works somewhat decently :)

Here we demonstrate the ability to “breed” ensembles by using their total predictions as a model and making F2-F3 offspring generations. Something we didn’t do was add up/compare the predicted class probabilities instead, and this was due to the difficulty in combining Neural Networks (that output probabilities) and classifiers like SVM, also due to Excel’s easy usage.

A Note on Inbreeding: In an attempt to achieve an accuracy close to our top-performing models (while spot-checking what they got wrong) with the limited amount of Kaggle submissions we had and no dedicated testbench, we gave a lot of influence to these high-performing CNNs, so much so that the ensemble just converged to the predictions of the CNNs, with only a few different answers per ensemble. → **GENETIC DIVERSITY IS IMPORTANT!**

Also, had we had a dedicated testbench, an option would have been to write a Genetic Algorithm to do this “model breeding” for us, and simply take the most accurate result(s).

FINAL THOUGHTS

Our Feelings on the Matter (LOL) And Kaggle Leaderboard

The final models we chose for Kaggle submission were the iteration of the **W+MFCC Attention Network** that had the highest public Kaggle accuracy (actually the one with BatchNorm layers still turned on for training mode and not off for eval mode), and we let the other model be whichever among the highest accuracy models Kaggle decided to pick for us.

Turns out we should've listened to PyTorch's "put your model in eval mode or risk inconsistent predictions" warning, as our eval mode model was tied for first place at **80.9%** private accuracy, yet had lower public accuracy, causing it to not be auto-picked. Our highest private accuracy that was picked was **80.09%**, which came from one of our CNN Ensembles (of course, with the Attention Network as well).

Overall we explored a lot and had a lot of fun, could've gotten higher had we optimized our hyperparameters more, as we largely did exploration and less tuning. We won in our hearts :)

Miniaturization of Large Architectures

State-of-the-Art models often involve convolutional multiscale analysis and attention, resulting in VERY deep (or wide) networks. A primary goal of our group, which we also believe to be a very valuable skill and interesting field of research, was to somehow keep the functionality of these large architectures while downsizing them to our 4-convolution limit.

The way we approached that with our WiggleNet Attention Network was to implement the "**Wiggle Filter**", the 5x31 convolution filter that served a **dual purpose**: both to capture large parts of the MFCC "shapes" at once, and to downsample the image into something more manageable for the smaller convolutions. For our convolutions, we then implemented differently-sized filters for multiscale analysis, and created 2 parallel branches of such convolutions, allowing for analysis on different levels of detail as well as the ability to learn a suitable downsampling filter for one of our residual connections.

Similarly, while we didn't use our Multimodal CNN for this purpose, the intention behind it was also to split the wavelet decomposition of 1D data into sections, onto which resolution-dependently-sized convolutions would be applied, paying attention to the level of detail that mattered for each band of frequency.

Pitfalls/Shortcomings and Solution Recap

- No testbench/data leakage:
    - Developed mental heuristic to know what to expect.
    - Not wasting Kaggle submissions and using that to test.
    - Comparing similarity in the Ensemble Spreadsheet.
    - Not having testbench meant we had more training data - could even reshuffle train/val split to get the model to see all data points.
- Did not do a comprehensive runthrough of every possible model - may have missed good model/feature combinations

    - This was a natural, we ended up trying a lot of things so it did work out (kinda?), although if we had time, more pre-planning would not hurt
    - However, if we had the time, there are so many more things we would like to try. Maybe Hilbert-Huang and a better Attention Network will take the crown?
- Sometimes Colab would crash and models would fail to download, losing us the files
    - Smarter use of RAM stopped the crashing. Saving the predictions onto a spreadsheet did at least let us make simple majority votes even with the pytorch file gone.
- Inability to spend too much time to tune hyperparameters
    - We would say this one was fine, because our exploration of new features and techniques was not in vain, and also we had a ton of fun :)
- At one point MLPs would get stuck giving only the same prediction for every input. To be honest, we were not sure why that happened, but we assumed that we tried feeding it all 20 MFCCs instead of 13 using an inappropriate network size for the dimensionality. We simply tuned the model architecture until it worked.
- Mistakes in coding models/understanding of feature extraction led us to do things we didn’t intentionally want to do at times.
    - Solution: None, but the things we did on accident kind of worked out.

Methods we would like to try next

(like a "next steps" section, if we had the time and resources)

- In hindsight, perhaps KNN on t-SNE data would not be so bad?
- Vector Quantization of MFCCs, trying actual LPCCs
- Enforcing the first layer of an MLP/CNN to have a certain Kernel shape: (RBF, Gabor Filter, etc.learned from a different technique such as SVM, etc.)
- Coding an automatic learning rate scheduler
- Allowing Hilbert-Huang to run for longer to achieve higher resolution, and doing analysis on a 3D tensor where the 3rd dimension is the dimension of IMFs instead of summing across this dimension to arrive at a 2D image.
- Using more MFCC-esque features as well as other features
- Using 1st and 2nd derivatives of MFCC features
- Implementing a Genetic Algorithm for model breeding
- Combining prediction probabilities into ensemble method
- Tuning wavelet choice and parameters / attention network params/hyperparams
- Splitting the data on lower-level classifications before moving onto the full 8 emotions
- Try more complicated multi-step training regimes

Cool Takeaways:

- Learning rate scheduling is very important for both convergence and generalization (thanks, 1cycle & co.!)
- Saving models to warm-startup helps if you have Colab usage limits.

- Measuring the timing of different models was useful to predict how long it would take in the future, allowing better scheduling, returning to the to avoid Colab disconnecting you after it's done fitting.
- There was another Huang who authored many papers on Hilbert-Huang, who was not the one who published the original paper. He improved upon the original technique, was from France, presented his work at the British Royal Society of Science, and had ties to the Chinese government. The original Huang was a NASA engineer and did none of the things above.
- Ensembles are great for scaring others in competitions in early stages, as the early accuracy increase looks like your group's progress is deceptively fast. Just make sure you have a good gene pool for it.
- Attention networks are good. Wiggle. Wiggle them large convolutions.
- Explore. Read a lot. Y'know.
- Convolution sizes are important. Don't use max pooling; it literally is subsampling (THE BAD ONE where you ignore values). DON'T subsample, it removes information. Recently literature has abandoned local max pooling for global average pooling and subsequent convolutions.
- Cool features are good. Wavelets and Hilbert Huang. Strong features allow you to distinguish data better. Bad features make good models irrelevant.
- Despite popular code readability beliefs, reuse variables, don't overload precious RAM.
- Write code in a modular way. I.e. put neural networks in a block, easy to edit elements in the block, swap blocks.
- First iteration of the attention network didn't converge, that was because downsampling was too harsh. Be careful of how you downsample.
- MMMMMultiscaaaaleeeeeeeeeeeee…!.
- Pre-emphasis.

CODEBASE

Unfinished Gender Detector:
https://colab.research.google.com/drive/1IVQXSEeo7vXnvqQo-OpSn00ZhtXdO50J?usp=sharing
Messy Main Colab Doc:
https://colab.research.google.com/drive/1pMYaAacWtx-oSbXZwJqCb5_ppXNXjckD?usp=sharing
Excel Ensemble With Possibly Incorrect Metric Formulas:
https://docs.google.com/spreadsheets/d/1Ku_ZNxsjJp2QsUST0Uq-Xcn_4b7wt0NdMPkQXbmPXhA/edit#gid=1795901463

ACKNOWLEDGEMENTS

- Jeff, for being so kind to us in terms of deadlines and help.
- Vishwa, for offering us guidance, putting up with our questions, and entertaining every hypothetical we had.
- Rosemary, for putting up with a constant torrent of Discord messages and questions, as well as linguistic discussions and constant encouragement.
- And the following civilians for helping us manually label gender data: Jimmy Lin, Shen Zhang, Lunea Wang, Brian Baskin, Liam Waite, Maggie Ku, David Zhao, Eli Blasini, Kausar Alkaderi, Shihab Sarar Ahmed.
- Everyone else, for putting up a friendly competition atmosphere.

REFERENCES

Project: https://www.kaggle.com/competitions/elec-301-speech-emotion-classification/overview
PyTorch Documentation: https://pytorch.org/
EMD Package: https://emd.readthedocs.io/en/stable/index.html
Librosa: https://librosa.org/
NumPy: https://numpy.org/doc/stable/index.html
Pandas: https://pandas.pydata.org/docs/
Fast-TSNE: https://github.com/huguyuehuhu/fastTSNE
Scikit-Learn: https://scikit-learn.org/stable/
Scipy: https://scipy.org/
Matplotlib: https://matplotlib.org/stable/index.html
PyWavelets: https://pywavelets.readthedocs.io/en/latest/ref/index.html
OpenSmile: https://audeering.github.io/opensmile-python/usage.html#audformat

TIMNet: https://arxiv.org/pdf/2211.08233v1.pdf
Hilbert-Based MFCC Features:
https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5571052
Jitter and Shimmer: https://wiki.aalto.fi/display/ITSP/Jitter+and+shimmer
CNNs for Time-Series: https://arxiv.org/pdf/1611.06455.pdf
Wavelet MLP: https://personal.ntu.edu.sg/elpwang/PDF_web/00_icci.pdf
Study on CNN Receptive Fields (and how they are often too small):
https://www.cs.toronto.edu/~wenjie/papers/nips16/top.pdf
1x1 convolution and its uses:
https://machinelearningmastery.com/introduction-to-1x1-convolutions-to-reduce-the-complexity-of-convolutional-neural-networks/

Gender-Driven Speech Emotion Recognition:
https://www.researchgate.net/publication/258023891_Gender-Driven_Emotion_Recognition_Through_Speech_Signals_For_Ambient_Intelligence_Applications
Semantic Segmentation of Images w/ Global Convolutional Network:
https://arxiv.org/pdf/1703.02719v1.pdf
The All-Convolutional Net (on max pooling being bad): https://arxiv.org/pdf/1412.6806.pdf
Attention-Augmented Convolutions: https://arxiv.org/pdf/1904.09925.pdf
Adam vs SGD:
https://medium.com/geekculture/a-2021-guide-to-improving-cnns-optimizers-adam-vs-sgd-495848ac6008
Batch size in relation to generalizability:
https://machinelearningmastery.com/how-to-control-the-speed-and-stability-of-training-neural-networks-with-gradient-descent-batch-size/#:~:text=Smaller%20batch%20sizes%20are%20used,i.e.%20when%20using%20a%20GPU).
Batch size effects on training:
https://medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e#:~:text=large%20batch%20size%20means%20the,all%20about%20the%20same%20size.
Ghost-Batch Normalization (generalization comes from the number of parameter updates alone): https://arxiv.org/pdf/1705.08741.pdf
Don't Decay LR, Increase Batch Size: https://arxiv.org/pdf/1711.00489.pdf
Val Loss vs Training Time vs Batch Size:
https://wandb.ai/ayush-thakur/dl-question-bank/reports/What-s-the-Optimal-Batch-Size-to-Train-a-Neural-Network---VmlldzoyMDkyNDU
Dynamic Loss Functions:
https://proceedings.neurips.cc/paper/2018/file/8051a3c40561002834e59d566b7430cf-Paper.pdf
Spatial Attention for CNNs: https://arxiv.org/pdf/2108.08205.pdf
1-cycle/Super-Convergence w/ Large Batches & LR: https://arxiv.org/abs/1708.07120
Wavelet CNNs: https://arxiv.org/abs/1805.08620
Wavelet-Based MFCC-like features for Speech Emotion Recognition:
https://www.sciencedirect.com/science/article/pii/S2090447916301514
More Wavelet CNNs:
https://arxiv.org/abs/2005.03337#:~:text=Our%20experimental%20results%20on%20ImageNet,robustness%20than%20their%20vanilla%20versions.
Large Kernel Design in CNNs: https://arxiv.org/abs/2203.06717
Global-Aware Multiscale CNN for Speech Emotion Recognition:
https://arxiv.org/pdf/2204.05571.pdf
Aspects of Consonants:
https://corpus.eduhk.hk/english_pronunciation/index.php/3-2-acoustic-aspects-of-consonants/
Popular Tutorial Solutions:
https://towardsdatascience.com/speech-emotion-recognition-using-ravdess-audio-dataset-ce19d162690
https://github.com/rudrajikadra/Speech-Emotion-Recognition-using-Librosa-library-and-MLPClassifier

https://towardsdatascience.com/speech-emotion-recognition-with-convolution-neural-network-1e6bb7130ce3

At one point we decided to label the sentences by what was being said to supervised-ly train a classifier to sort them: we didn’t end up doing it, but this was the speech-to-text model we used to expedite the manual labeling process.
https://github.com/snakers4/silero-models
(technically this is a pre-trained model being used to manually label data, LOL; however, this labeling wasn’t actually the true labels of the clips, and we had Jeff’s permission to manually label data with non-emotion labels :))