Yun Jia Zhuang & Janelle Tam & Sina Salmannia Feb 15th, 2024

MAIS 202 - Preliminary Results

1. Problem Statement

We want to train a model to automatically classify the emotion of a given piece of audio data, into one of the following categories: neutral, calm, happy, sad, angry, fearful, disgust, surprise.

2. Data Preprocessing

We will be working with the following datasets:

- Surrey Audio-Visual Expressed Emotion (SAVEE) (sample size = 479)
  - anger, disgust, fear, happiness, sadness, surprise, and neutral
- RAVDESS Emotional speech audio (sample size = 1439)
  - calm, happy, sad, angry, fear, surprise, and disgust
- Toronto emotional speech set (TESS) (sample size = 2799)
  - anger, disgust, fear, happiness, pleasant surprise, sadness, and neutral
- CREMA-D (sample size = 7441)
  - anger, disgust, fear, happiness, neutral, and sad

Common emotions amongst all: anger, disgust, fear, happy, sad

We chose not to work with IEMOCAP and EmoReact datasets anymore, since we were unable to get the dataset files in time.

Current preprocessing methods

- We are currently not deleting any data, and will try training with the entire corpus. We may come back to delete some data.
- We chose to normalize all input data in order to account for the difference in recording conditions between different datasets (decibels, power, etc.).
- We are ensuring that all files are of the same length, namely **2 seconds**. Longer files are truncated, while shorter files are padded with 0's.
- We will be converting each audio file to its MFCC, delta and delta-delta features. The concatenation of these features will be fed to our model as input. We have chosen to use these features, as MFCCs are commonly used for this classification task, and adding delta (velocity) and delta-delta (acceleration) features help to "recognize speech better".

3. Machine learning model

We will be using a CNN for this classification task, since from our research, we have seen that it is the most frequently used model for emotion recognition based on audio.

a) We will be using **PyTorch** to implement our CNN model. We are also using the Librosa library for audio processing (see Data processing section), and other libraries such as scikit-learn and numpy.

   We will be using two convolution layers with max pooling and 20% dropout, and three linear layers. Here is the detailed architecture of our model:

```
    (0): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
         ceil_mode=False)
    (3): Dropout(p=0.2, inplace=False)
    (4): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (5): ReLU()
    (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
         ceil_mode=False)
    (7): Dropout(p=0.2, inplace=False)
    (8): Flatten(start_dim=1, end_dim=-1)
    (9): Linear(in_features=13248, out_features=256, bias=True)
    (10): ReLU()
    (11): Linear(in_features=256, out_features=128, bias=True)
    (12): ReLU()
    (13): Linear(in_features=128, out_features=8, bias=True)
    (14): Softmax(dim=1)
```

Here are two different visualizations of our model's architecture.

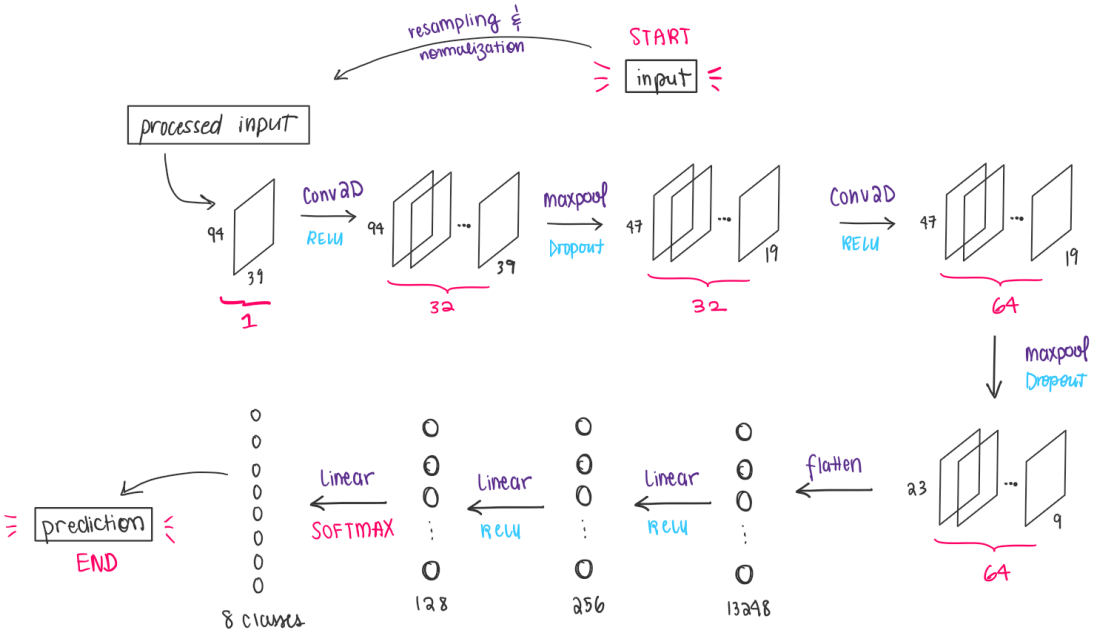

Fig. 1: Visualization of model architecture, with sizes indicated.

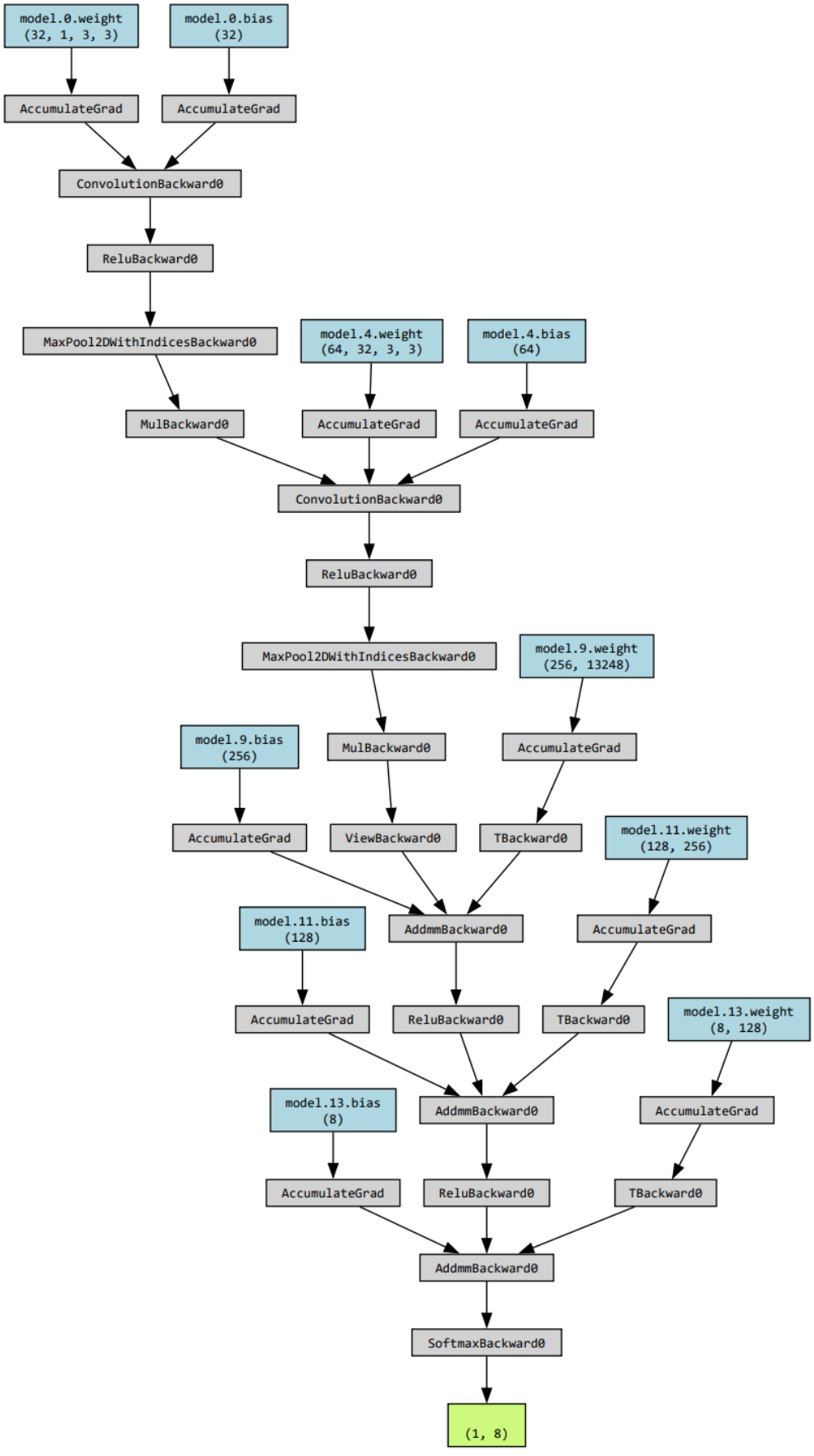

Fig. 2: Visualization of model architecture using PyTorchViz.

b)

- We have chosen an 80-20 train test split, as it is a common split.
- We are using k-fold cross validation (k=4) over the training data, after testing it out with a few different folds. Online sources show that k=5 and k=10 folds are typically good choices, but for runtime constraints we have chosen k=4.
- We are using a 20% dropout rate between convolutional layers for regularization.
- For hyperparameters, we are performing a grid search over the learning rate (0.01, 0.005, 0.001, 0.0005).

c) The performance of the model depends on the learning rate used.
For learning rates 0.0005 and 0.001, our model is consistently overfitting across all 4 folds, for each learning rate. There is a ~15-20% difference in accuracy between testing and validation accuracy (depending on the fold and the learning rate), for a best validation accuracy of 61%.
For learning rate = 0.005, we are neither overfitting nor underfitting.
For learning rate = 0.01, we are severely underfitting; our model is not learning anything.

d) We had trouble in feeding the input to the model, specifically in terms of making sure that the dimensions were consistent with the model's layers' expected inputs.
To solve this, we searched for solutions online and used ChatGPT to track down the errors; the most prominent fix was to unsqueeze input sizes.

We had also noticed that the implementation of k-fold cross validation was not done properly, as the training accuracy would increase between each fold. After careful examination, we realized that our model was not reinitialized between folds, so the training was leaking.

4. Preliminary results

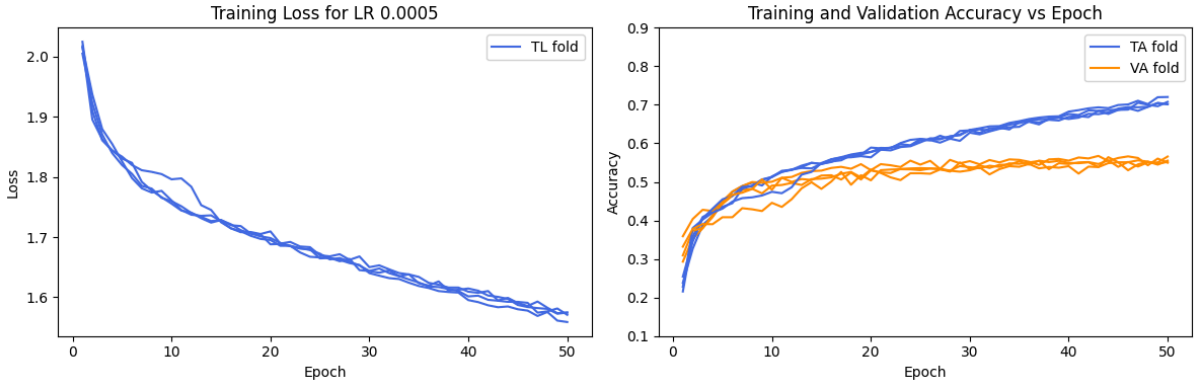

Fig. 3: Training loss and training/validation accuracies over all folds (k=4), lr=0.0005. Training accuracy steadily increases and reaches 70%, while validation accuracy remains stagnant at around 55%. This indicates overfitting.

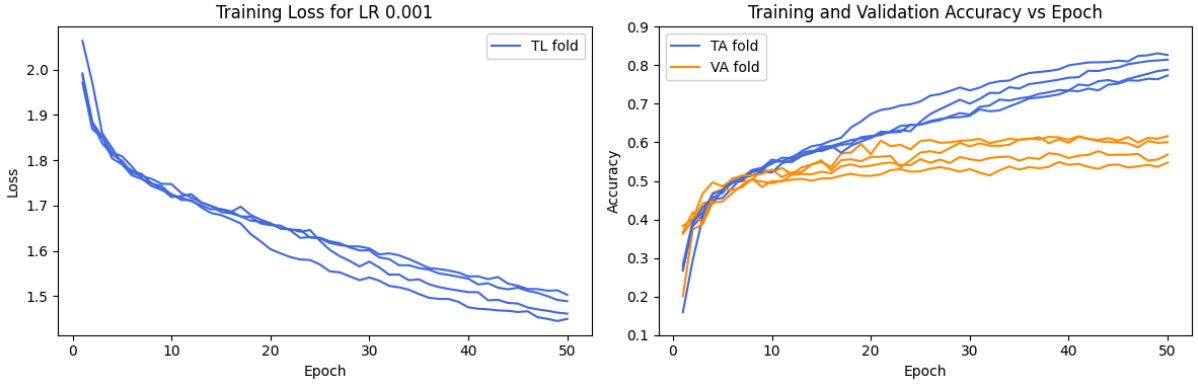

Fig. 4: Training loss and training/validation accuracies over all folds (k=4), lr=0.001. Training accuracy steadily increases and reaches 80%, while validation accuracy remains stagnant at around 52-61%. This indicates overfitting.

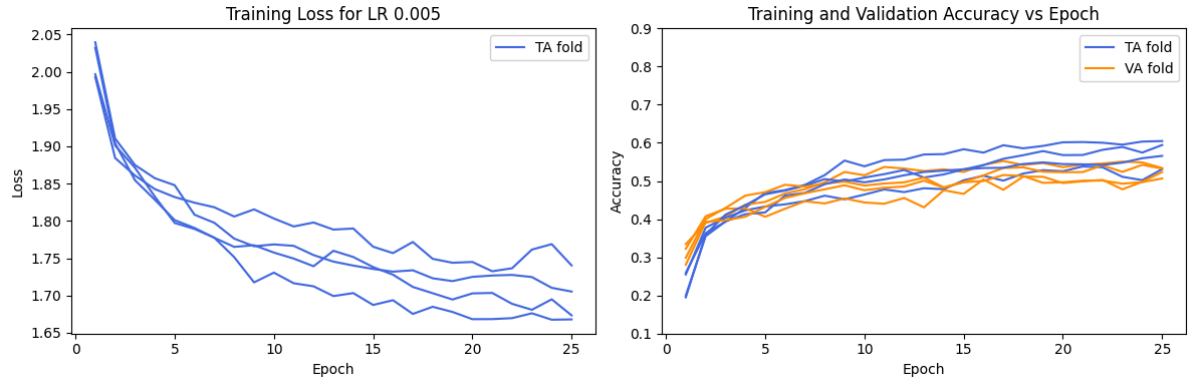

Fig. 5: Training loss and training/validation accuracies over all folds (k=4), lr=0.005. Training and validation accuracy are similar across all folds, over all epochs, with validation accuracies hovering around 50%. Accuracy might increase with more epochs, but it doesn't seem computationally feasible to increase the number of epochs, as accuracy increases very slowly.

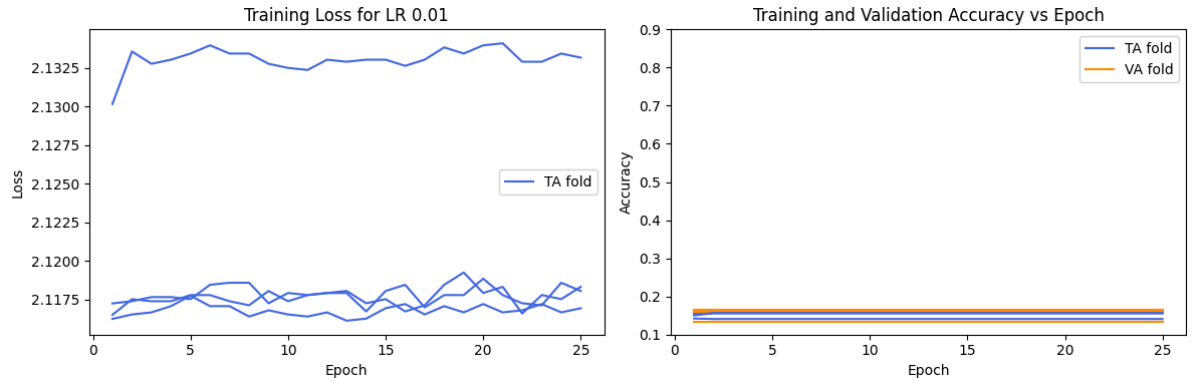

Fig. 6: Training loss and training/validation accuracies over all folds (k=4), lr=0.01. Training and validation accuracies are close to nil. This is a bad learning rate, and the model is not learning anything.

Through k-fold cross validation, we can see that our model is consistently overfitting across all folds, for lr=0.0005, 0.001. It collapses for lr=0.01, and is decent but slow for lr=0.005.

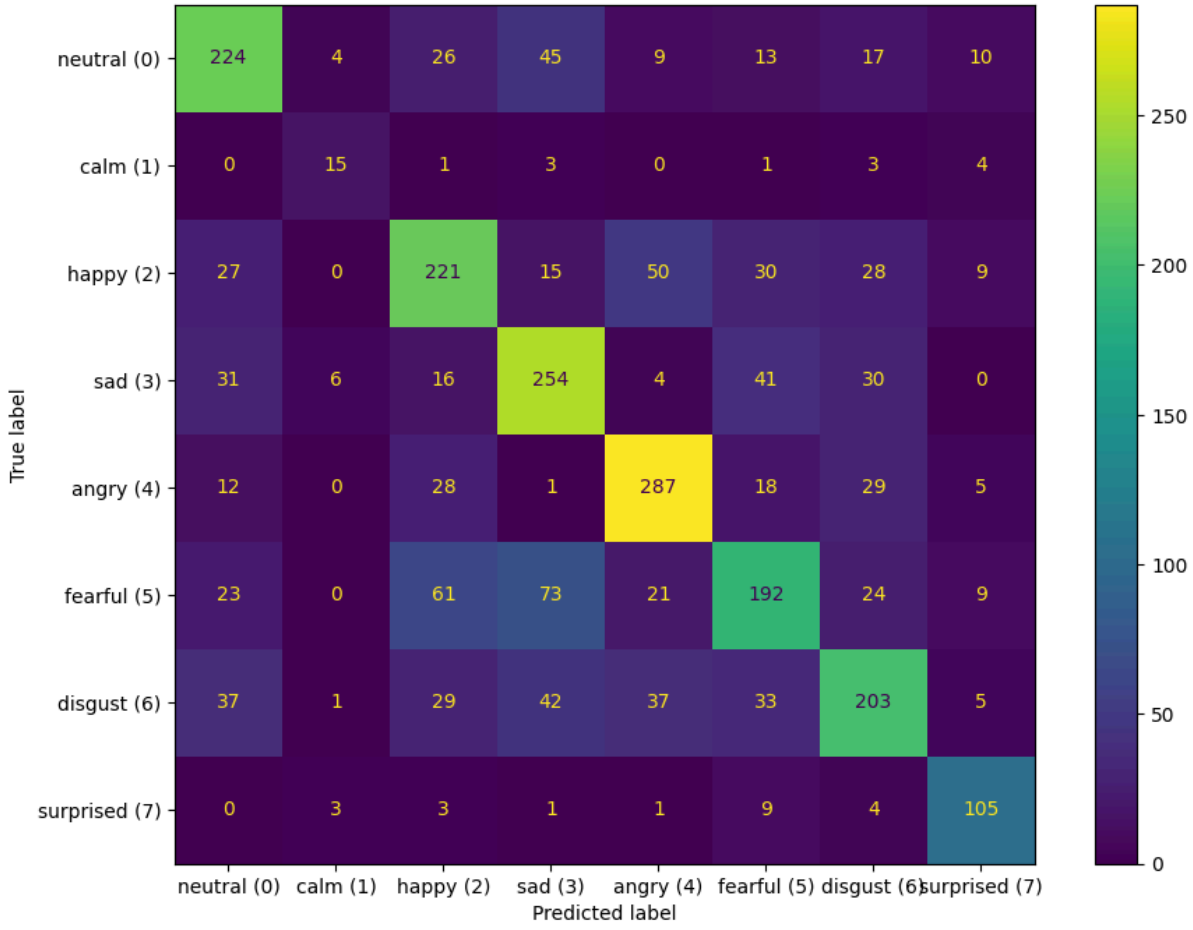

Fig. 7: Confusion matrix on test dataset. We can see that the model's accuracy in predicting "calm" audio files is very low. In retrospect, this is due to the low sample size for calm data, and we should have omitted "calm"-labelled data from our dataset.

Summary
Overall, our model (taking lr=0.001) performs decently well, and we have a test accuracy of **61.69%** in a classification task over 8 categories for a test dataset of size 2433, which is pretty decent.
We will, however, still be working on improving our model to see if we can decrease overfitting.

5. Next steps

Since the input to our CNN model must be consistent, we have concerns about how test audio (from outside of our dataset) would be processed to be fed to our CNN model. More specifically, recording conditions of external audio, and buffer (starting and ending) silence segments in the audio may cause issues if we simply truncate the audio to 2 seconds. So we will be looking into alternatives to crop out silence from audio.

We were also interested in allowing longer audio inputs (up to 4s), but the only concern is that our training data is not long to begin with. We will look into a way to pad or loop our training data to 4s, and see if that works.

We will also look into other datasets that could be used for our model, and see if that improves training accuracy.