- Start with results, then go in order of rubric
- Next meeting: wednesday July 29, 8 pm Toronto time (finish script (prelim) by then)
- Record video on saturday, 10 am
- Finish slides / finalize script by saturday
- Talk about final report on saturday
- * we need to email the prof by friday if there's a particular time slot that we can't make for the presentations
- Future:
    - Try RNN
    - Try some other embedding state
    -
    - Better transfer learning model to get encodings from
    -
    -

Hi Everyone , we're Team 11.

Problem (Reynold)
Imagine if you're sitting in a concert far away from the stage and you wanted to know what instruments are playing.
This is the problem that our group decided to solve with machine learning is the problem of music instrument classification.This problem is interesting since it begs the question about what defines an instrument.Our research into existing problems shows that many programs are able to identify music instruments individually. The novel aspect of our program is that it will not only attempt to identify one instrument by itself but multiple instruments playing together. This is the application of developing a machine learning model that can identify features from a mixed audio file which we are proposing through a convolutional neural network.

(next slide)
We separated our project into two stages, one in identifying single solo instruments, and then a second stage to identify which instruments are present. We scaled down the problem to include just two instruments in each sample.

Demonstration

-

Quantitative Results (Reynold)

For our baseline model, we decided to go with a random forest model. A random forest model creates a bunch of random decision trees that based on the individual results of each decision tree, it classifies accordingly. This was the model chosen because we wanted to see if the random forest can find the features that define an instrument. Results are compared 1D convolutional neural

Comparing our stages of the model. One can see from the table comparison in the validation accuracy of the baseline model compared to our actual convolutional network model. (show It's interesting to note that there are features that CNN can learn much better than our random forest model.

- Create a visual showing the baseline model vs our model
- Introduce random forest

Baseline:

15.1 %, 38% - 100
13.8%,38.7% - 500
14.3%, 37.3% - 50
13.8% , 38.7 - 200

Data:

We used IRMAS dataset for our training data. This is a dataset that contains solo music played by 11 different types of instruments across multiple genres and styles, including country-folk, classical, pop-rock and latin soul with potential drum beats in some samples. The audio is represented in .wav format and each audio file contains 2 channels since they were recorded as stereo sound. All audio files contain a 3s music and were recorded with 44100Hz sample rate. Thus the sample data will be input to the model as tensors of size [2, 3 * 44100]. From this dataset we picked 4 instrument types: acoustic guitar, piano, trumpet and violin for their distinct sound features. And this table shows the number of samples in each class. This is a reasonable amount of data to fit our training capacity in Google Colab and achieve considerably good generalization.

Data Processing:

Since our classes are not balanced, we first added augmented data to the classes with fewer samples in order to balance data for all types of instruments. The data augmentation process was done by reversing signs in some samples to create new data. Each balanced dataset contains 721 samples in total, split into 60% training, 20% validation and 20% test set. Then we normalized our data within the range of -0.5 to 0.5. In this way we made sure the volume of our audios fall into the same range, which would be significant when we mix audio samples to create multi-instrument audios, because we don't want one instrument to be loud enough to shadow another. The augmented and normalized data are used in the network model in stage 1, where only single instrument audios are required. For stage 2 training data, we created dual-instrument audio files by summing samples in each two classes and then normalized the data between -0.5 to 0.5. In the end we got 6 dual-instrument classes and 721 samples in each class. We didn't hit for triple instruments or more, because the problem of distinguishing them becomes much harder for both humans and neural network models. The source data has some other limitations too. Besides having only 4 instruments and only 2 classes in combined audios, the source data have limited music styles and the combined audios may not resemble the real-world duets, since each instrument in the combined audios has the normalized volume but in real life, there may be a predominant instrument with much higher volume than others. Additionally,the instruments in the combined audios may not even play the same music. This case may not happen in real duets.

Model

For both stages of our project, we constructed one-dimensional CNNs.

<u>"Stage 1: Instrument Classification" slide</u>

In the first stage, we trained a model to output a four-dimensional vector representing the probability across the four instrument classes. **(transition)** The training labels are one-hot encoding vectors identifying the true class the sample belongs to. **(transition)** And cross entropy loss was used during training. .

<u>"Stage 1: Tuning the Model" slide</u>

(transition) When tuning the stage 1 hyperparameters, we found that there were some hyperparameters that significantly improved performance. This included: **(one transition per point)**

- *Adding dropout layer*
- *Reducing parameters in the fully connected layer*
- *Increasing convolution layers and size of filters per layer*

(transition) We also experimented with other hyperparameters and **(transition)** found the best combination shown here, with optimal validation accuracy reached within 30 epochs.

<u>"Stage 2: Multi-Instrument Classification" slide</u>

In the second stage, we trained a model to identify the instruments **(transition)** present in data samples. The output of our model **(transition)** is a four-dimensional multi-label classification vector, where each value represents the probability of the corresponding class being positive. This is achieved by applying the sigmoid activation to the output vector. **(transition)** The target vector is a multi-hot encoding vector indicating the two instruments that are actually present in the sample. A major difference between this stage and the previous is that **(transition)** the multi-label soft margin loss is used in order to allow for multiple positive labels. This loss function also accounts for the sigmoid activation.

<u>"Stage 2: Training the Model" slide</u>

The training process for Stage 2 was very similar to Stage 1, **(transition)** with numerous hyperparameter combinations tested. Coincidentally, we found that the optimal hyperparameters and architecture for Stage 2 are the same as those of Stage 1, but with optimal validation accuracy reached within 20 epochs instead of 30. This interesting finding prompted us to explore transfer learning, which we will discuss later.

<u>“1D CNN Architecture” slide</u>
The final architecture for both stages has a total of 6 convolution layers. **(transition)** Due to the large input size, we alternate each convolution layer with max pooling of various sizes to obtain a tensor of manageable size for the fully connected layer. **(transition)** The first three convolution layers have 64 filters, and gradually decreasing kernel sizes from 11 to 7, while the last three layers **(transition)** gradually decrease the number of filters and maintain a kernel size of 5. **(transition)** We flatten the features into a vector for the fully connected layer, and obtain a four-dimensional output for the labels.

<u>“Stage 2: Transfer Learning Architecture” slide</u>
We then explored transfer learning to examine the difference in performance between the two models. **(transition)** Using the best weights from the convolutional layers of stage 1, we obtained feature representations of the audio inputs. These features were then flattened and used as inputs to a fully connected multi-label classifier, as shown here. After tuning the network, we found the best hyperparameters to include a learning rate of 0.0001, batch size of 64, and optimal validation accuracy to be reached within 10 epochs.

<u>“Stage 2: CNN vs. Transfer Learning” Slide</u>
In terms of computation time and memory required, the transfer learning model has a clear edge. Transfer learning would allow us to train less than half the number of parameters. Both the computation time per epoch as well as overall computation time are significantly less with transfer learning. From these metrics, we can see that there is a huge potential with this concept. This could be further explored as the focus of future projects.

Qualitative Results (Tiffany)

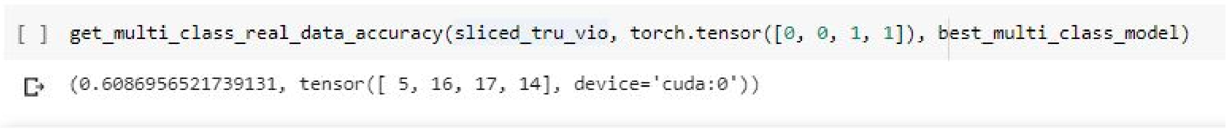
<u>“Summary of Results” Slide</u>
Here is a summary of the validation and test results from both stages. We find that in both stages, our models significantly outperform the baseline. We can see from the stage 2 results that the transfer learning validation accuracy is comparable with that of the initial CNN. However, the test accuracy differs by 5%. This reflects some limitations in the transfer learning model’s ability to generalize to new data. **(transition)** Since the model with the higher validation accuracy in Stage 2 is the 1D CNN, we will focus on this model in the results discussion moving forward.

<u>“Stage 1 Test Results” slide</u>

Breaking down the test results of stage 1 **(transition)**, we can see that the overall test accuracy is not evenly represented across all classes. **(transition)** The trumpet class immediately stands out to be the best, with the highest true positive rate along with lowest false positive and negative rates. However, the guitar class, **(transition)** which has the second highest true positive rate, also has the highest false positive rate. Additionally, **(transition)** the piano and violin classes account for almost all of the guitar's false positives *and* negatives. Considering the numerous styles of playing the guitar, piano, and violin, these results are reasonable. For instance, picking guitar strings or playing pizzicato notes on the violin could both sound similar to piano notes when played individually. This idea of similarity between these instruments is further evidenced by **(transition)** the piano's false negatives, which are mostly misclassified as guitar samples.

<u>"Stage 2 (CNN) Test Results" slide</u>
With the second stage, the accuracy for individual binary classifications was close to 70%. **(transition)** However, looking at the confusion matrix, we can see that when it came to classifying all four binary labels for each sample, there are some interesting results. Apart from the fact that around half the multi-label predictions classified an incorrect number of instruments (as shown in the rightmost column), we can see a pattern that segregates the multi-label classes into two parts -- **(transition)** -- a group for the samples containing the guitar, and another group for the samples without the guitar. Within each subgroup of multi-label classes, there are high false positive and negative rates. It appears that the network has trouble differentiating between instrument pairs within the same subgroup. Additionally, **(transition)** misclassifications outside of those subgroups are rare in comparison. This observation suggests that it is much easier to detect the presence of the guitar than it is to detect the presence of the piano, violin, or trumpet, when in a mixed setting. Perhaps in the future, experimentation could be done with more instruments or other instrument combinations to see whether this pattern occurs with other instruments.

Demo
(discuss about performance on real duets)


```
[ ]  get_multi_class_real_data_accuracy(sliced_tru_vio, torch.tensor([0, 0, 1, 1]), best_multi_class_model)

     (0.6086956521739131, tensor([ 5, 16, 17, 14], device='cuda:0'))
```

(the music (trumpet + violin duet) contains 23 [2, 132299] slices:). Total accuracy rate is 0.608696%

Now back to the concert scenario. If you've been listening to the background music so far, you would realize that it is a duet composition of a trumpet and a violin. We saw previously that the test accuracy on the holdout dataset is 69.74%, but how does our model perform on this background music? When in a real life scenario, it appears that there is a gap in performance, with 60.86% accuracy. Considering the synthetic attributes and disadvantages of the combined data used to train the model, it makes sense why this performance is lower. Out of the 23 samples, our model identified the presence of the trumpet in 17 and violin in 14 of the slices, with many piano and minimal guitar false positives. This further illustrates the analysis of our stage 2 model.

~~Although this is a reasonable performance, we can see many false positives in the piano class. However, considering the minimal false positives in the guitar class, the difficulty of differentiating between the piano, trumpet, and violin in a mixed setting is shown, while reflecting how much easier it is to differentiate between the absence and presence of the guitar.~~

Takeaways & future improvements (Ahmed)

- The problem was harder than we thought: the poor results of the baseline models is an indicator that the problem is complex. Additionally, the inputs are very large given the number of frames and harder to process before training.
- In Stage 2, we wrote the training function with BCEWithLogitsLoss() which was inconvenient when implementing multi-label classification. It would be better for future machine learning problems to search for implemented functions in library documentation such as Pytorch.
- Since machine learning is dependent on many libraries which are updated frequently these days, it's a good idea to keep track of library versions you are working with. For instance torchaudio library was updated mid-project which worked with a later version of pytorch. This caused major issues with GPU drivers. Hence, we had to keep torchaudio to its older version since Google Collab's GPU drivers work with an older version of pytorch.
- For future improvements, we could do 3-instrument multi-label classification.