Name: Nhat Minh Dang

ID: 222172836

SIT 374: My Sprint 2 task:

A) Introduction:

Convolutional Neural Networks (CNNs) are used commonly for audio data processing, sound event detection and classification task. For Weather Event Detection, I will digest deeper in two significant Weather Events, Ice Storm and Tornado.

B) Implementation:

1. Data Collection:

- Because the Data File is too large to push, here is my link for Drive upload:

https://drive.google.com/drive/folders/13p8viUYT0cLJkCS4QAjZ1RVzbg_tl4CS?usp=sharing

- Most of my data is collected from FreeSound. Here are some of the significant records.
- For IceStorm:
  - IceStorm3: Live Recording of Ice Storm in Portland.
  - IceStorm4: 1-hour Live Recording at Metro Detroit.
  - IceStorm7: 6 hours of Recoding outside environment.
  - IceStorm8: Live Recording of Ice Storm and Pine Straw.
- For Tornado:
  - Tornado1: An example of small tornado with hail.
  - Tornado3: Live Recording Tornado at Oklahoma.
  - Tornado4: Air Recording at Bundeswehr.

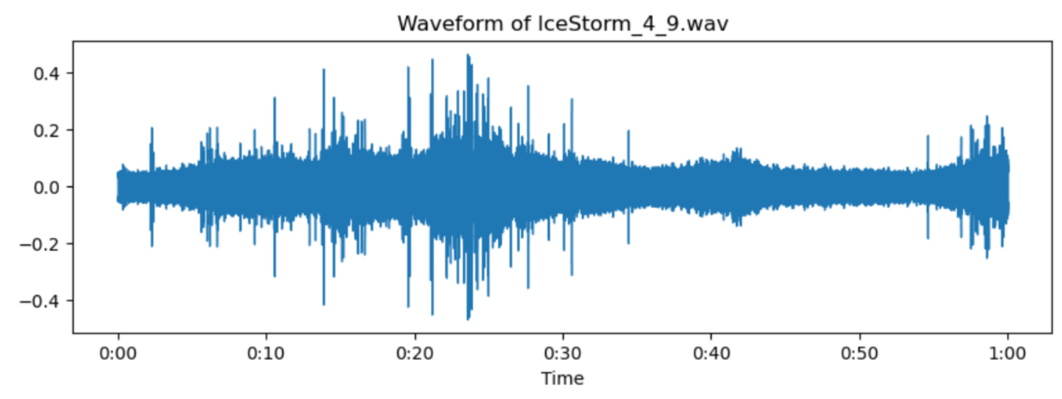
2. Library use:

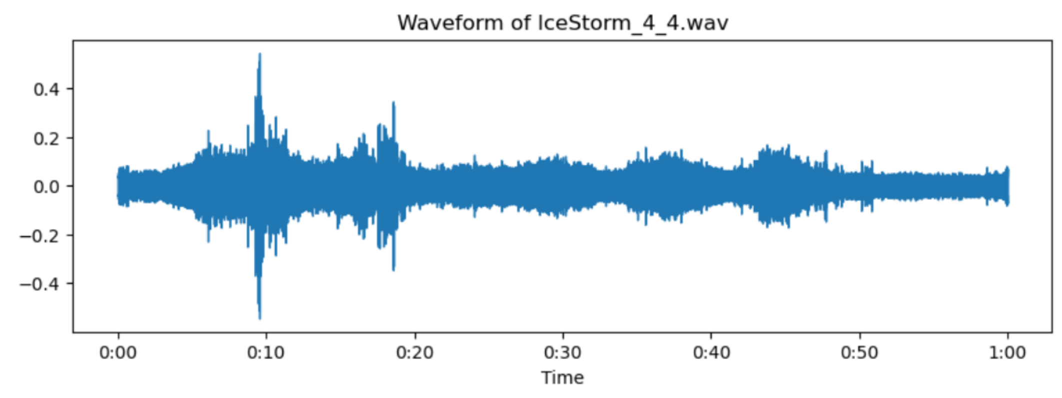
These are the main libraries:

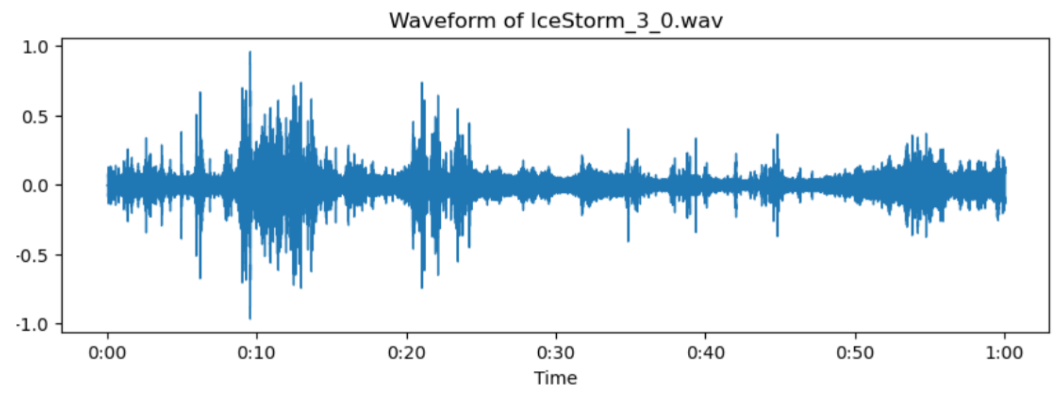
- Librosa: Process Audio input.
- Mathplotlib: Visualize result.
- Sklearn and tensorflow: Train data using Neural Network.

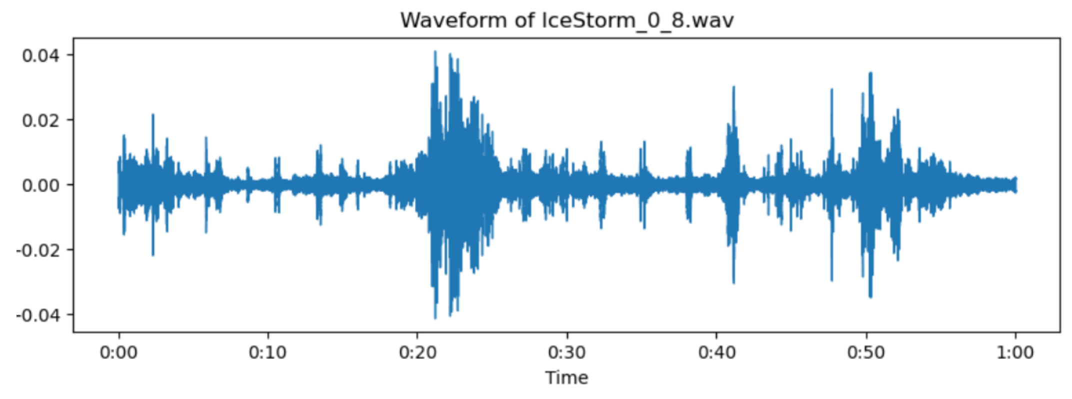
3. Load audio files and process:

After loading the audio inputs, I divide the audio into small chunks with 1 minute each. Next, I visualize the audio Wave form as plots. Below picture is my example of randomly selected chunks after breaking. To increase the generalization of model, I have added audio data using Augmentation. By adding more data that is modified into different angle, the models can detect more variety of audio and reduce the chance of overfitting. The process includes adding white noise, time stretching and pitch shifting. In addition, I also process the audio inputs to fit into a specific shape before fitting into model









4. Training using Neural Network:

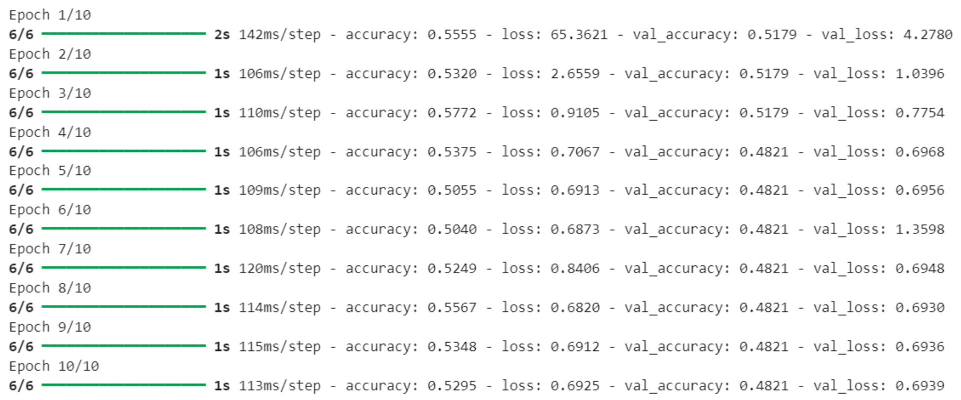
These are the layers that I used in this CNN:

- Input: My input shape is (128, 128, 1).

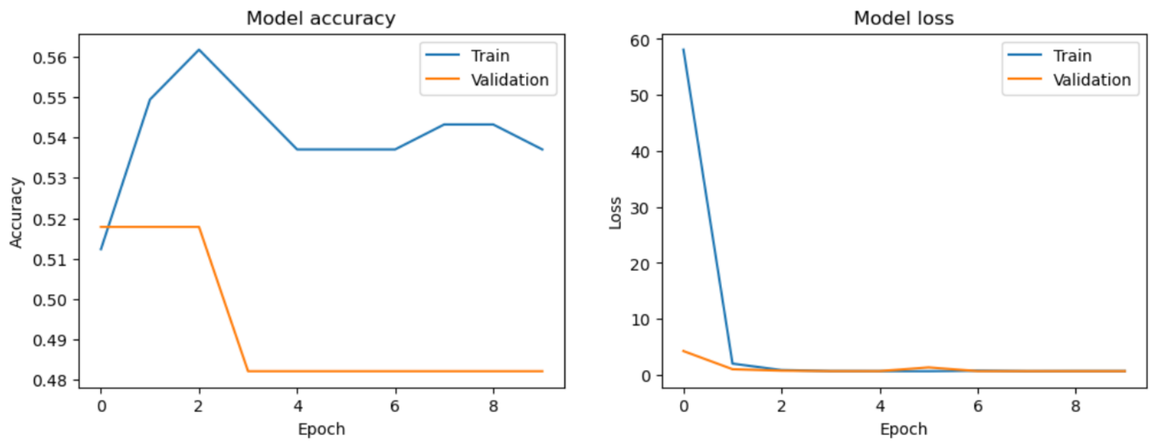
- First Convolutional Layer: Using ReLu and applies 32 filters of size (3, 3) to extract low level features.
- First Max Pooling Layer: Down samples by taking the maximum value from each (2, 2) window.
- Second Convolutional Layer: Using ReLu and applies 64 filters of size (3, 3) to extract more complex features.
- Second Max Pooling Layer: Down samples by taking the maximum value from each (2, 2) window.
- Third Convolutional Layer: Using ReLu and applies 128 filters of size (3, 3) to extract even more complex features.
- Third Max Pooling Layer: Down samples by taking the maximum value from each (2, 2) window, furthermore, reducing the spatial dimension.
- Flatten Layer: Reshapes the 3D feature maps into 1D vectors before feeding into Dense layer.
- Two Dense Layer: The first one use ReLu to learn high level patterns while the second one use Softmax to convert the output into probabilities.

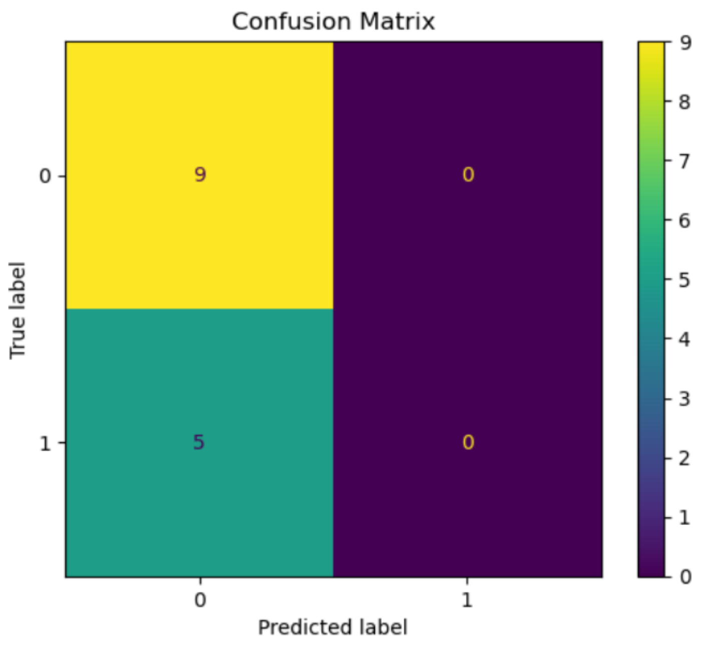
C) Result:

- These are all the maximum result I achieved from both Event Detections. For evaluation, I use accuracy and loss ratio, followed by confusion matrix and classification report.
- For IceStorm:



```
Epoch 1/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 2s 142ms/step - accuracy: 0.5555 - loss: 65.3621 - val_accuracy: 0.5179 - val_loss: 4.2780
Epoch 2/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 106ms/step - accuracy: 0.5320 - loss: 2.6559 - val_accuracy: 0.5179 - val_loss: 1.0396
Epoch 3/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 110ms/step - accuracy: 0.5772 - loss: 0.9105 - val_accuracy: 0.5179 - val_loss: 0.7754
Epoch 4/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 106ms/step - accuracy: 0.5375 - loss: 0.7067 - val_accuracy: 0.4821 - val_loss: 0.6968
Epoch 5/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 109ms/step - accuracy: 0.5055 - loss: 0.6913 - val_accuracy: 0.4821 - val_loss: 0.6956
Epoch 6/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 108ms/step - accuracy: 0.5040 - loss: 0.6873 - val_accuracy: 0.4821 - val_loss: 1.3598
Epoch 7/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 120ms/step - accuracy: 0.5249 - loss: 0.8406 - val_accuracy: 0.4821 - val_loss: 0.6948
Epoch 8/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 114ms/step - accuracy: 0.5567 - loss: 0.6820 - val_accuracy: 0.4821 - val_loss: 0.6930
Epoch 9/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 115ms/step - accuracy: 0.5348 - loss: 0.6912 - val_accuracy: 0.4821 - val_loss: 0.6936
Epoch 10/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 113ms/step - accuracy: 0.5295 - loss: 0.6925 - val_accuracy: 0.4821 - val_loss: 0.6939
```



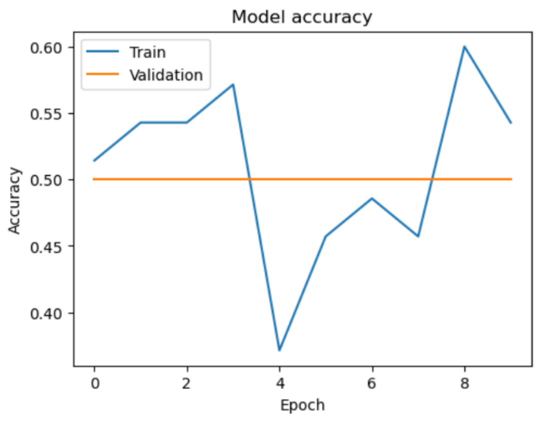


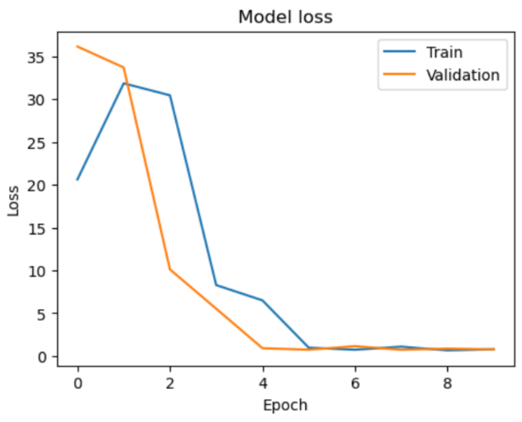
```
]: print(classification_report(y_test, y_pred, target_names=['Class 0', 'Class 1']))
```

```
              precision    recall  f1-score   support

     Class 0       0.64      1.00      0.78         9
     Class 1       0.00      0.00      0.00         5

    accuracy                           0.64        14
   macro avg       0.32      0.50      0.39        14
weighted avg       0.41      0.64      0.50        14
```

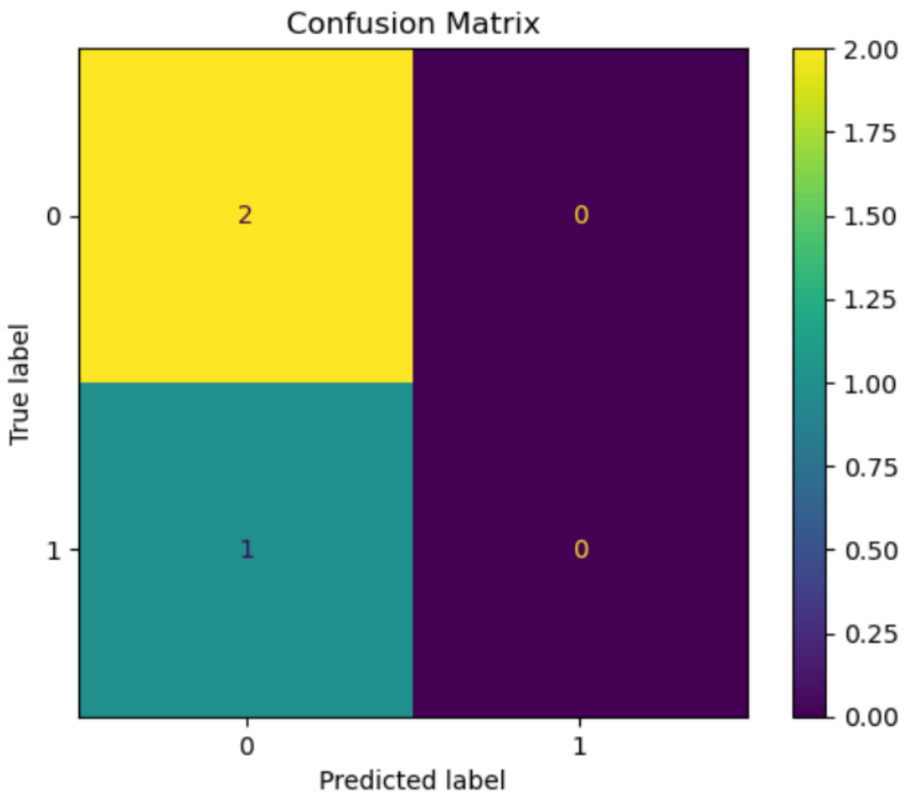
- For Tornado:

```
Epoch 1/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 2s 238ms/step - accuracy: 0.5199 - loss: 14.5152 - val_accuracy: 0.5000 - val_loss: 36.1498
Epoch 2/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 91ms/step - accuracy: 0.5286 - loss: 32.8477 - val_accuracy: 0.5000 - val_loss: 33.7031
Epoch 3/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step - accuracy: 0.5286 - loss: 31.4050 - val_accuracy: 0.5000 - val_loss: 10.1199
Epoch 4/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 88ms/step - accuracy: 0.5685 - loss: 8.4317 - val_accuracy: 0.5000 - val_loss: 5.5288
Epoch 5/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 79ms/step - accuracy: 0.3830 - loss: 6.4952 - val_accuracy: 0.5000 - val_loss: 0.9137
Epoch 6/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step - accuracy: 0.4506 - loss: 0.9866 - val_accuracy: 0.5000 - val_loss: 0.7366
Epoch 7/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step - accuracy: 0.4801 - loss: 0.7361 - val_accuracy: 0.5000 - val_loss: 1.1494
Epoch 8/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 86ms/step - accuracy: 0.4714 - loss: 1.0874 - val_accuracy: 0.5000 - val_loss: 0.7285
Epoch 9/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 80ms/step - accuracy: 0.5979 - loss: 0.6585 - val_accuracy: 0.5000 - val_loss: 0.8668
Epoch 10/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 75ms/step - accuracy: 0.5286 - loss: 0.7939 - val_accuracy: 0.5000 - val_loss: 0.7722
```





```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 111ms/step
```



```
print(classification_report(y_test, y_pred, target_names=['Class 0', 'Class 1']))
```

```
              precision    recall  f1-score   support

     Class 0       0.67      1.00      0.80         2
     Class 1       0.00      0.00      0.00         1

    accuracy                           0.67         3
   macro avg       0.33      0.50      0.40         3
weighted avg       0.44      0.67      0.53         3
```