ECGR 4127 – Machine Learning for IoT
Christian Martens – SID#801143623
Prof. Holleman

Project 2 Report – Keyword Spotting

Team Members: Christian Martens

Introduction:

In this project, the goal was to create a machine learning model which could classify and distinguish between two activation words, random words, and silence. The team needed to create a dataset, build a custom TensorFlow model, train the model on the dataset, convert the trained model to a TFLite file, and then replace the keyword spotting model in the example code with the TFLite object detection model. In the upcoming sections, the team will elaborate on the model's architecture, how data and training was completed, the results of the training, the performance of the deployed model, as well as steps taken to improve the overall accuracy of the model.

For this project, the team's selected two words were "apricot" and the name "Marius". The word "apricot" was selected because the team believed that it would provide a better example of training the model due to its unique characteristics compared to other words in the training set, and the name "Marius" was chosen arbitrarily to compare the difference in how \a "proper" keyword (that is, one that was chosen due to special characteristics of the word such as how "apricot" was chosen) might increase the accuracy of the model compared to a random keyword. Apricot started with the letter 'a', which none of the common words in the Google 30 dataset began with. Additionally, it is a tri-syllabic word which ends in an unusually harsh and pronounced 't' sound making it sound (at least, to human ears) very distinct and noticeable.

After testing the first model, the team was not impressed by the results. Although the model could classify the audio, it was very large and thus took too long to process. A number of models and datasets were made for this project to more accurately predict the chosen keywords based on deductions from the initial training models and datasets. These changes increased the accuracy of the model by a few percentage points.

Figure 1: A picture of apricots and a bust of the Greek General Gaius Marius

Model Architecture:

When classifying an audio sample/spectrogram, a Convolutional Neural Network (CNN) is a very practical choice. CNNs are very robust and efficient for their size which make them particularly enticing model designs for small micro-controllers such as the Nano. Due to this, the team's model uses this CNN structure. The first model the team created included 3 convolutional layers with batch normalization and 4 fully connected layers with rectified linear (ReLU) activation. The exact architecture for the model was created by adjusting the number of convolution layers, connected layers, and connected layer nodes until the model had an acceptable amount of parameters (under 50,000). The finalized model has 23,332 parameters. For this model, training time was not a factor in how many epochs were trained nor was it a significant limitation to the model's design. However, for future models, training time was significant due to the large constructed dataset so changes to those models were made to specifically accommodate a better training time.

Layer (type)	Output Shape	Param #
conv2d_23 (Conv2D)	(None, 24, 32, 2)	20
batch_normalization_24 (BatchNormalization)	(None, 24, 32, 2)	8
max_pooling2d_13 (MaxPooling2D)	(None, 12, 16, 2)	0
conv2d_24 (Conv2D)	(None, 12, 16, 4)	76
batch_normalization_25 (BatchNormalization)	(None, 12, 16, 4)	16
max_pooling2d_14 (MaxPooling2D)	(None, 6, 8, 4)	0
conv2d_25 (Conv2D)	(None, 6, 8, 8)	296
batch_normalization_26 (BatchNormalization)	(None, 6, 8, 8)	32
max_pooling2d_15 (MaxPooling2D)	(None, 3, 4, 8)	0
flatten_9 (Flatten)	(None, 96)	0
dropout_11 (Dropout)	(None, 96)	0
dense_23 (Dense)	(None, 128)	12416
dropout_12 (Dropout)	(None, 128)	0
dense_24 (Dense)	(None, 64)	8256
dropout_13 (Dropout)	(None, 64)	0
dense_25 (Dense)	(None, 32)	2080
dropout_14 (Dropout)	(None, 32)	0
dense_26 (Dense)	(None, 4)	132

Total params: 23,332
Trainable params: 23,304
Non-trainable params: 28

Figure 2: CNN Model #1's architecture and parameters

When pushing this model to the Nano, the team noticed a strange phenomenon. Despite its high validation accuracy, the deployed model nearly exclusively detected that the word ‘apricot’ was being called. After some consideration and studying of the resulting graphs, the team deduced that the dataset had two errors which seemed to be contributing to the deployment error. Firstly, the model had too few “silent” datapoints compared to the rest of the data. This was adjusted for in so the dataset had five times the number of silent datapoints. Secondly, the shift being applied to ‘apricot’ and ‘marius’ were, in some cases, completely removing the words from sound clip, resulting in an empty datapoint. The model would train on these empty keywords and, since there were many “silent” datapoints, the model began to learn that silence meant ‘apricot’. This was corrected by reducing the space between samples from 1/8 second increments to 1/16 second increments. For the second model, the team decided it was best to alter the model as opposed to creating a completely different architecture. The second model the team created included 2 convolutional layers with batch normalization and a single 4-by-4 max pooling in addition to 2 fully connected layers with rectified linear (ReLU) activation. The convolutional models had many more channels compared to the first model which significantly increased the number of parameters, but these parameters were nearly all from the resulting dense layers which are very quick for the model to calculate. This model was designed to test if a model with a small number of, albeit less complex, layers increased or decreased the accuracy compared to the original model. The exact architecture for the model was created by adjusting the number of convolution layers, connected layers, and connected layer nodes until the model failed to improve with the added complexity. The finalized model has 155,886 parameters.

Layer (type)	Output Shape	Param #
depthwise_conv2d_11 (DepthwiseConv2D)	(None, 24, 32, 1)	10
conv2d_18 (Conv2D)	(None, 24, 32, 64)	128
batch_normalization_18 (BatchNormalization)	(None, 24, 32, 64)	256
depthwise_conv2d_12 (DepthwiseConv2D)	(None, 24, 32, 64)	640
conv2d_19 (Conv2D)	(None, 6, 8, 96)	6240
max_pooling2d_11 (MaxPooling2D)	(None, 3, 4, 96)	0
flatten_7 (Flatten)	(None, 1152)	0
dense_19 (Dense)	(None, 128)	147584
batch_normalization_19 (BatchNormalization)	(None, 128)	512
dense_20 (Dense)	(None, 4)	516

Total params: 155,886
Trainable params: 155,502
Non-trainable params: 384

Figure 3: CNN Model #2's architecture and parameters

In addition to model #2, a replica model of model #1 was constructed using the updated dataset to give model #2 a fair comparison.

Layer (type)	Output Shape	Param #
conv2d_23 (Conv2D)	(None, 24, 32, 2)	20
batch_normalization_24 (BatchNormalization)	(None, 24, 32, 2)	8
max_pooling2d_13 (MaxPooling2D)	(None, 12, 16, 2)	0
conv2d_24 (Conv2D)	(None, 12, 16, 4)	76
batch_normalization_25 (BatchNormalization)	(None, 12, 16, 4)	16
max_pooling2d_14 (MaxPooling2D)	(None, 6, 8, 4)	0
conv2d_25 (Conv2D)	(None, 6, 8, 8)	296
batch_normalization_26 (BatchNormalization)	(None, 6, 8, 8)	32
max_pooling2d_15 (MaxPooling2D)	(None, 3, 4, 8)	0
flatten_9 (Flatten)	(None, 96)	0
dropout_11 (Dropout)	(None, 96)	0
dense_23 (Dense)	(None, 128)	12416
dropout_12 (Dropout)	(None, 128)	0
dense_24 (Dense)	(None, 64)	8256
dropout_13 (Dropout)	(None, 64)	0
dense_25 (Dense)	(None, 32)	2080
dropout_14 (Dropout)	(None, 32)	0
dense_26 (Dense)	(None, 4)	132

Total params: 23,332
Trainable params: 23,304
Non-trainable params: 28

Figure 4: CNN Model #3's architecture and parameters

Data and Training:

The data was collected in two ways. The first way was by simply recording ‘apricot’ and ‘Marius’ in a variety of tones, intonations, and pitches. Those base recordings were then subject to time-shifting, additional pitch manipulation, and noise integration to create a much larger dataset. The original datasets of roughly 50-60 samples of both keywords were turned into 15,000 samples of both keywords with varying pitch, timing, and noise level. This variety was added to ensure the model learned to recognize the keywords and not some other trait common among the data. To further increase the dataset size, the dataset was supplemented with synthetic data using A.I.-powered realistic voices from Speechify. Using synthetic data generation, an additional 150 base recordings of ‘apricot’ and ‘Marius’ were used created using a greater variety of pitches, tones, intonations, and inflections than the team could produce.

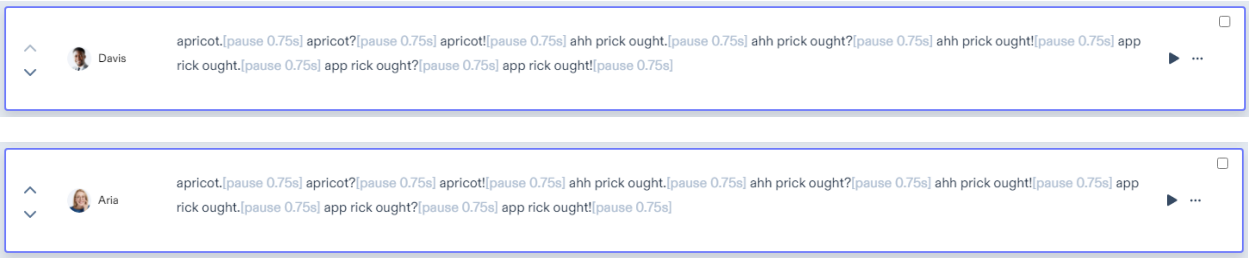

Figure 5: Pictures of the A.I. recording the various phrases of ‘apricot’ and ‘Marius’ on Speechify

As the dataset currently stands, the only thing the model can recognize is whether ‘apricot’ or ‘Marius’ is being said. However, those are not the only two possible states the model could find itself in. The model should be able to recognize silence and whether words other than the keyword are being said. Generating the silence dataset is easy as all we need to do is create a sufficiently large dataset of silence/noise. Creating the “filler” or “unknown” data is similar to how the keyword datasets were created. However, instead of using the same word, the filler data was constructed using over 100 different words of various syllables and phonemes. These 100 different words were chosen using two methods. The first method involved random word generation using all the letters of the alphabet as starting letters. The method was to use phonetic pangrams. Phonetic pangrams are sentences which use all the letters of a given language in them. These are especially useful in situations like these since pangrams tend to use sounds which aren’t very common in standard dialogue, so the model was able to learn more variety. The same models in Speechify were also used to generate some of this data to increase the variety of the dataset.

"That quick beige fox jumped in the air over each thin dog. Look out, I shout, for he's foiled you again, creating chaos."

"Are those shy Eurasian footwear, cowboy chaps, or jolly earthmoving headgear?"

"The hungry purple dinosaur ate the kind, zingy fox, the jabbering crab, and the mad whale and started vending and quacking."

"With tenure, Suzie'd have all the more leisure for yachting, but her publications are no good."

"Shaw, those twelve beige hooks are joined if I patch a young, gooey mouth."

"The beige hue on the waters of the loch impressed all, including the French queen, before she heard that symphony again, just as young Arthur wanted."

Figure 6: Examples of phonetic pangrams that the team used.

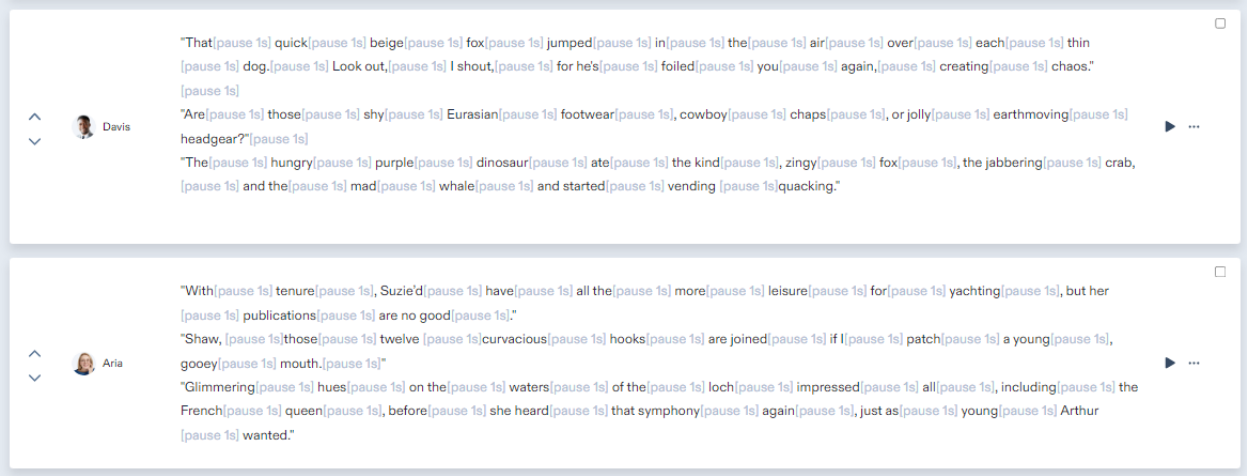

Figure 7: Speechify A.I. speaking the phonetic pangrams that the team used.

Results:

The validation accuracy of model 1 left was very high. With a 95.18% validation accuracy rating, one would expect that this model would perform very well. Once deployed, the model was sluggish due to its large size and nearly unerringly believed that 'apricot' was being constantly spoken. It took roughly 50 seconds to deploy the model onto the Nano and around 0.8 seconds to calculate a sample (a sampling rate of 1.2 samples/second). Primarily the model suffered from false positives. The confidence values for the keywords, silences, and unknown words were all very high and did not change much when a different sound was made. Due to

the complications with the first model's deployment, the second and third model were developed.

```
Epoch 10/10
2890/2890 [==============================] - 75s 26ms/step - loss: 0.2754 - accuracy: 0.8802 - val_loss: 0.1547 - val_accura
cy: 0.9518
```

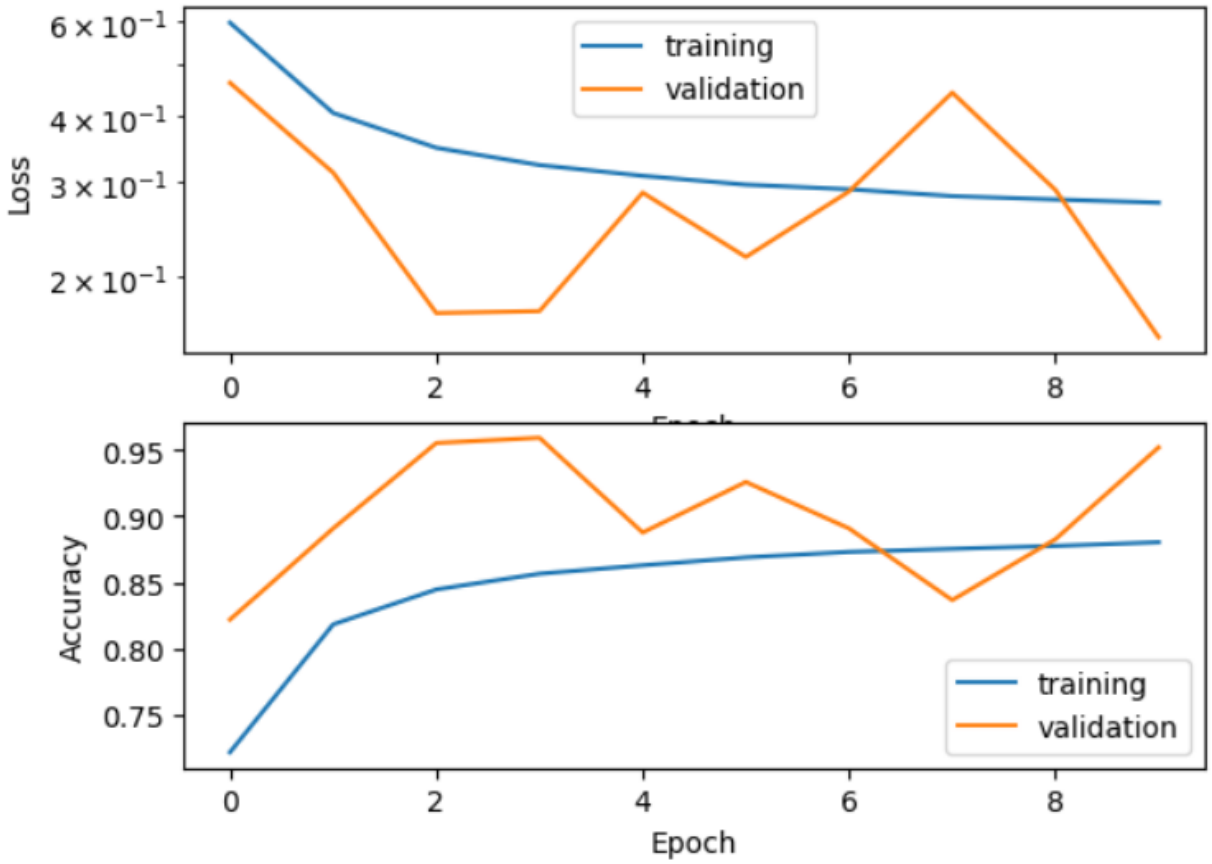

Figure 8: CNN Model #1's final loss and accuracy on training and test set with graphs.

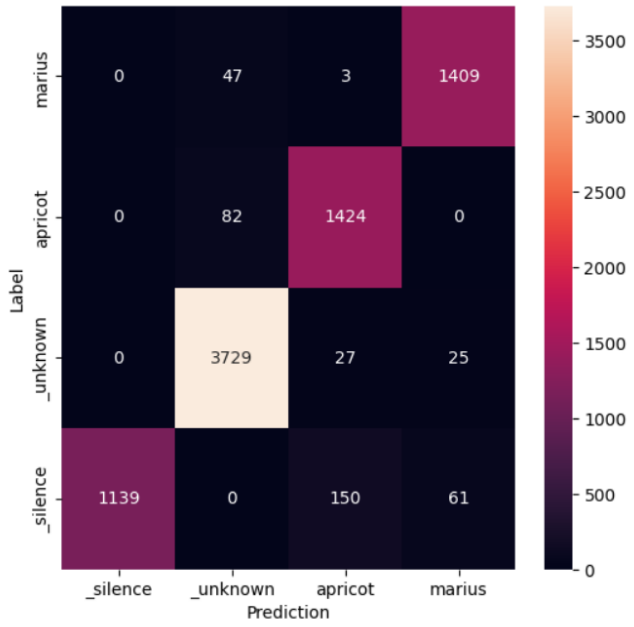

Figure 9: Validation Confusion Matrix for CNN Model #1

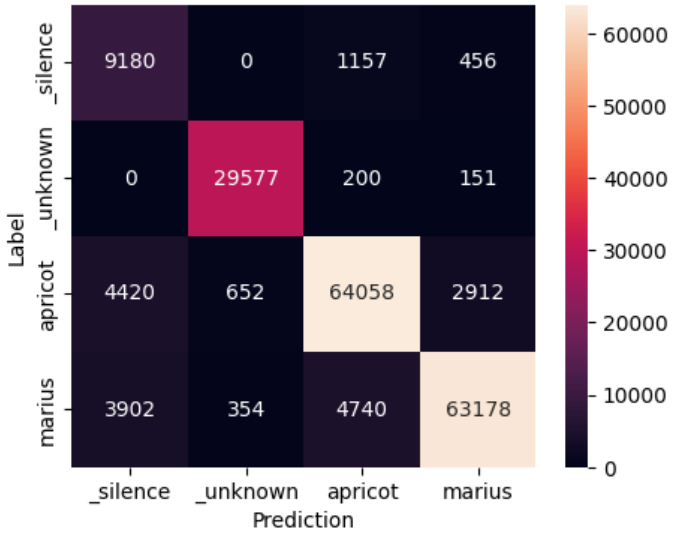

Figure 10: Training Confusion Matrix for CNN Model #1

The accuracy of model #2 was slightly higher than model #1 at 96.91% validation accuracy rating with a 0.1406 validation loss. Once deployed, the increased size of the model was made apparent due to the decreased sample rate. The second model took roughly 60 seconds to deploy onto the Nano and about 1.5 seconds to calculate a sample (a sampling rate of 0.67 seconds). Unfortunately, there were just as many false positives as with model #1. The

model, despite having a training accuracy of 93.20%, only successfully categorized 54% of the final training set.

```
Epoch 10/10
635/635 [==============================] - 182s 287ms/step - loss: 0.1755 - accuracy: 0.9320 - val_loss: 0.1406 - val_accura
cy: 0.9691
```

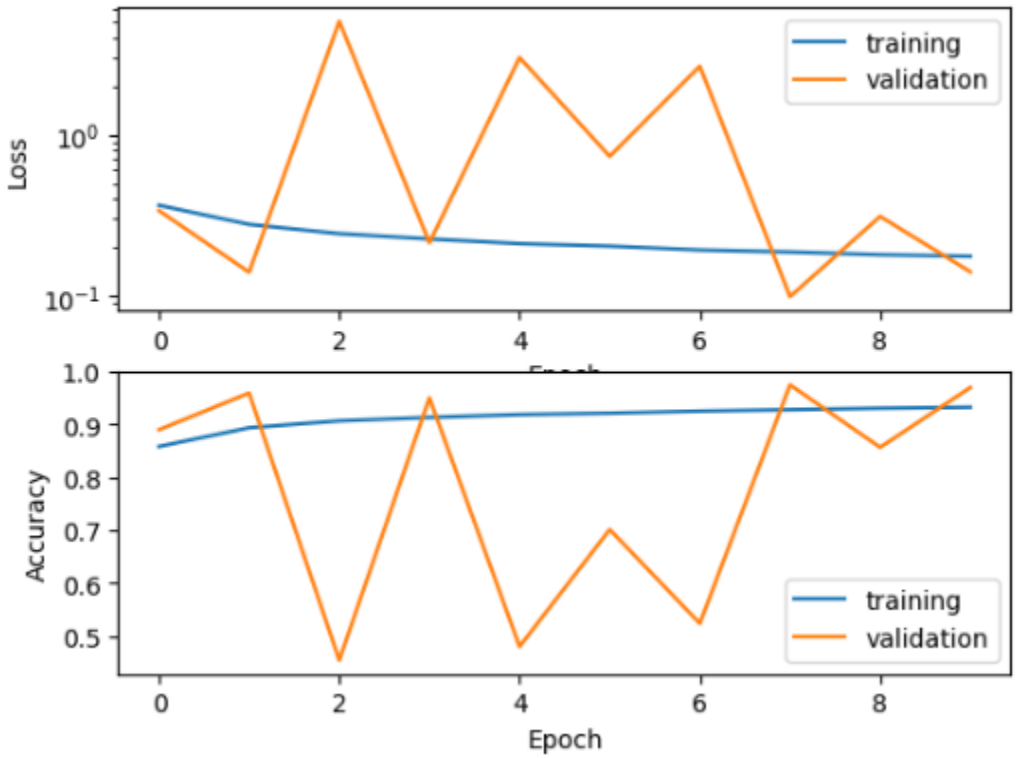

Figure 11: CNN Model #2's final loss and accuracy on training and test set with graphs.

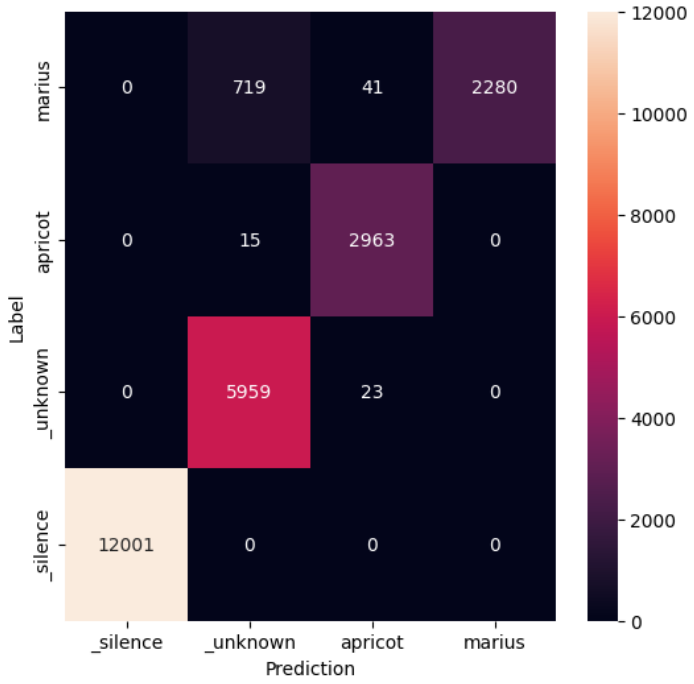

Figure 12: Validation Confusion Matrix for CNN Model #2

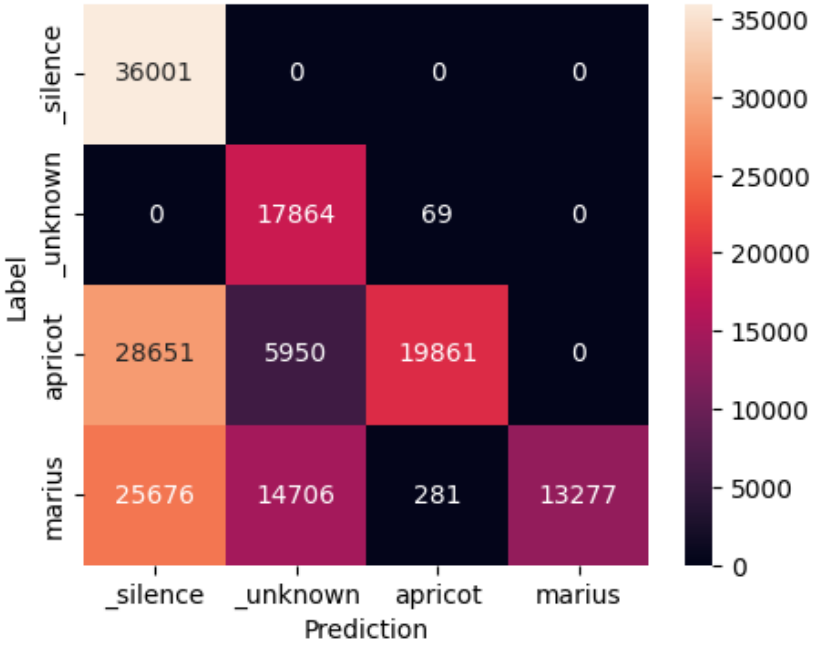

Figure 13: Training Confusion Matrix for CNN Model #2

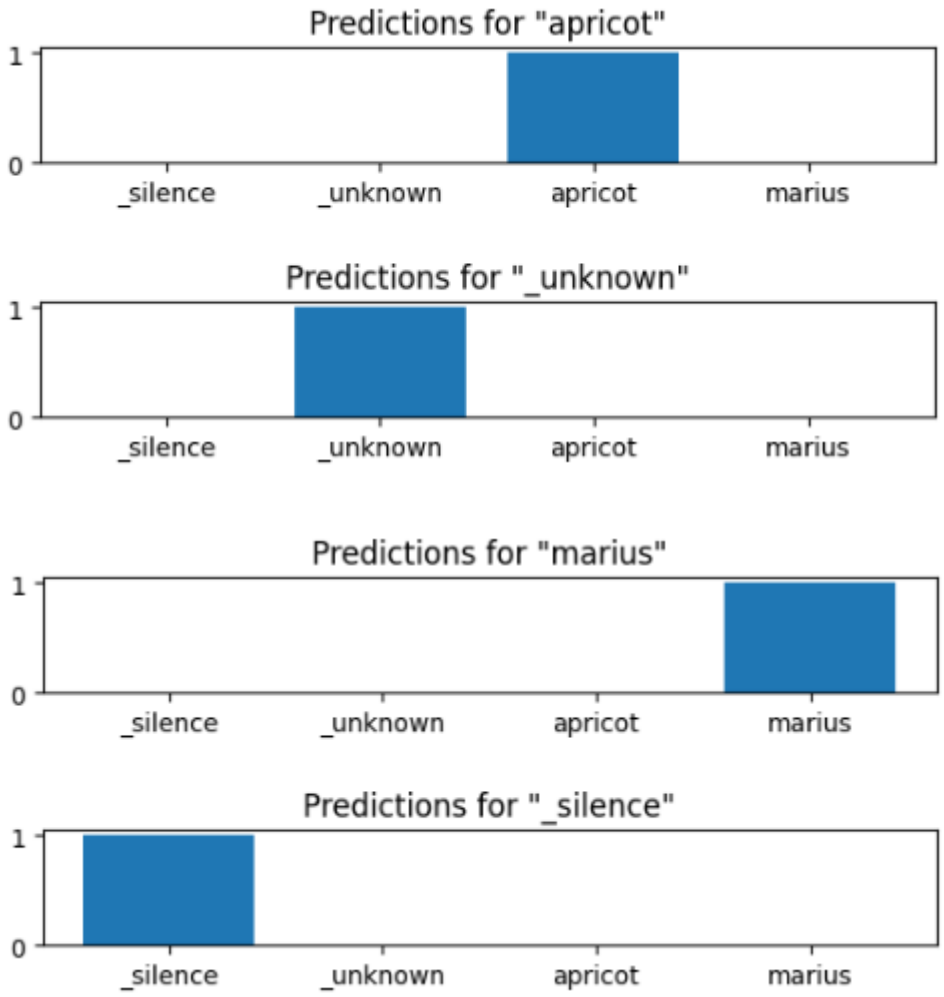

Figure 14: Predictions for Model #2

The accuracy of model #3 was slightly higher than model #1 or model #2 at 98.67% validation accuracy rating with a 0.0866 validation loss. Since the model was identical to model #1, the model had the same deployment time and sample rate. While there was still an issue with constantly predicting ‘apricot’, model #3 did seem less confident than with that calculation than model #1 and would recognize ‘Marius’ and unknown words occasionally.

```
Epoch 20/20
635/635 [==============================] - 46s 72ms/step - loss: 0.2345 - accuracy: 0.9083 - val_loss: 0.0866 - val_accurac
y: 0.9867
```

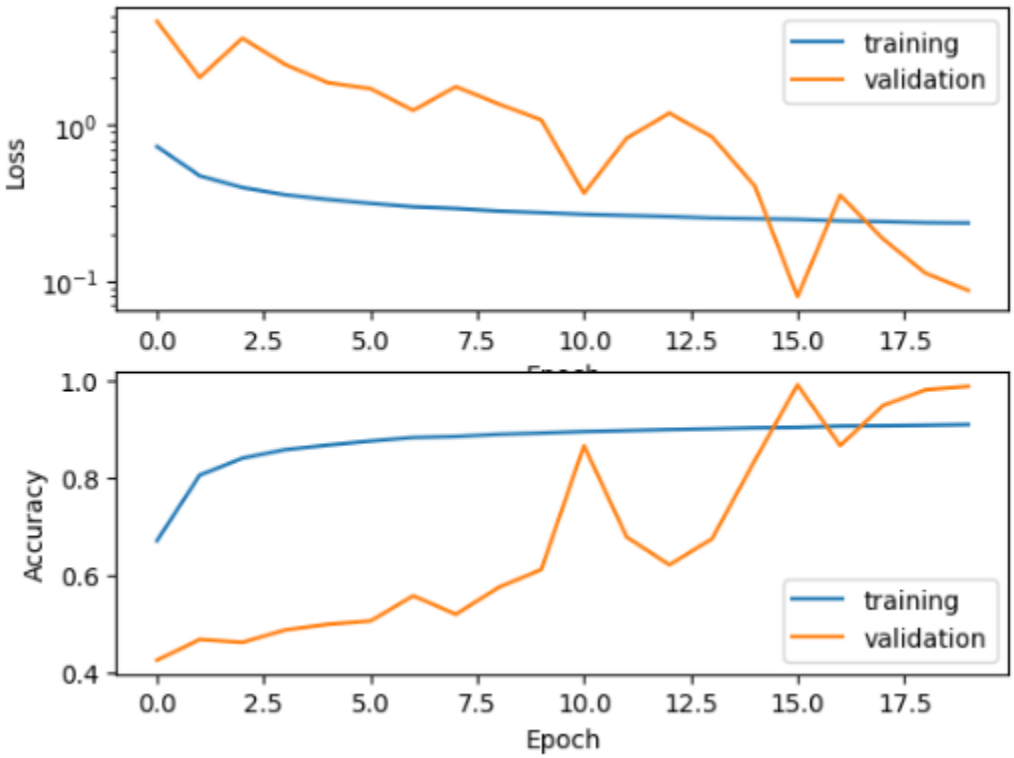

Figure 15: CNN Model #3's final loss and accuracy on training and test set with graphs.

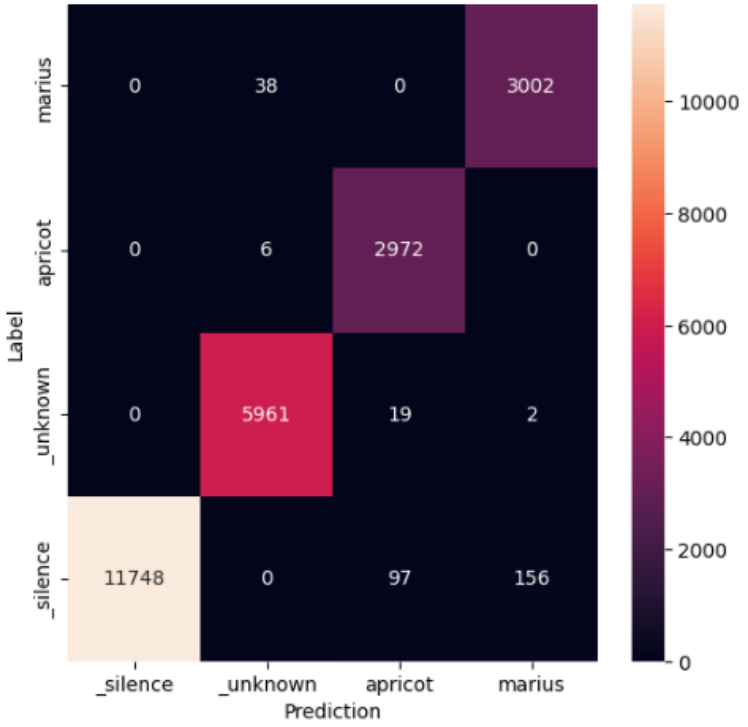

Figure 16: Validation Confusion Matrix for CNN Model #2

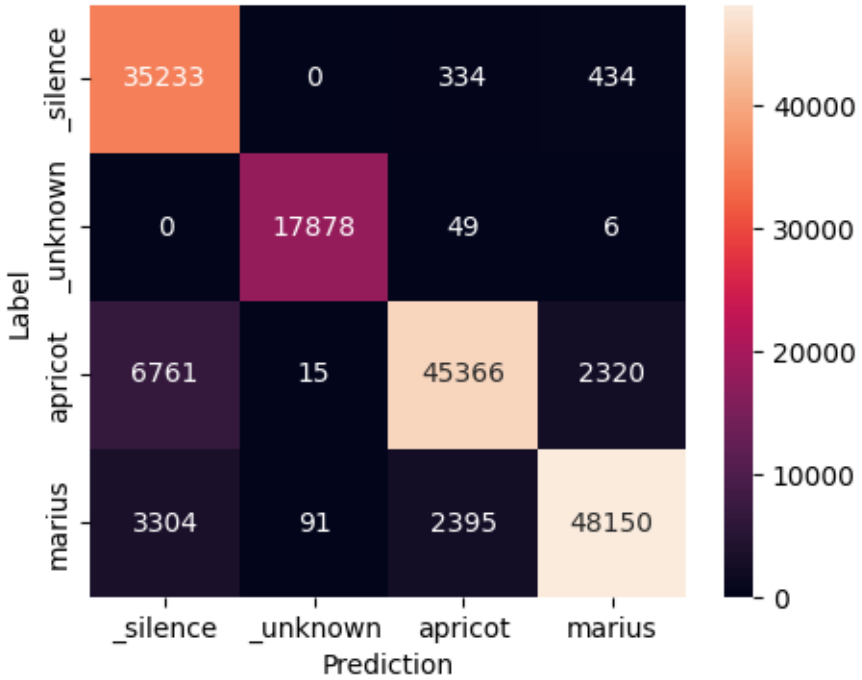

Figure 17: Training Confusion Matrix for CNN Model #2

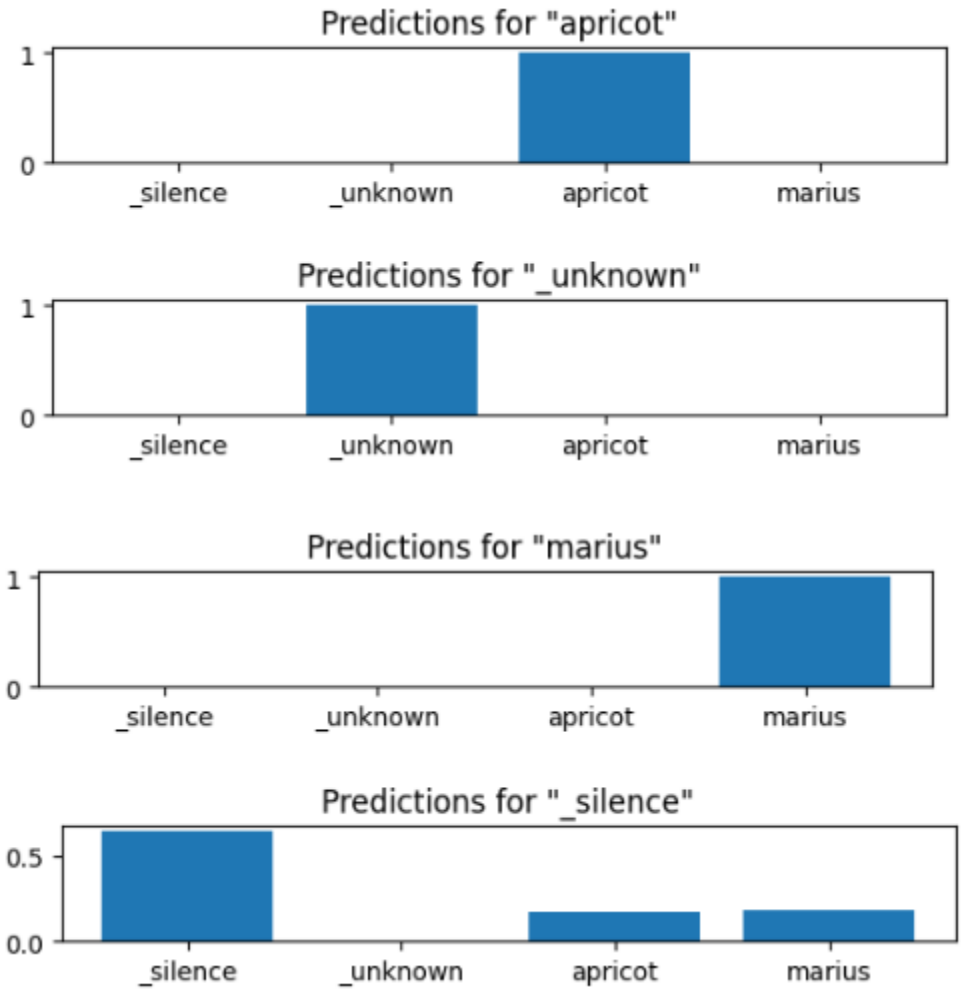

Figure 18: Predictions for Model #3

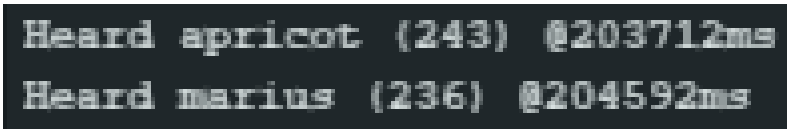

Figure 19: Deployed Model #3 recognizing keywords

Summary Table:

Model 1:

- Training Accuracy: 88.02%
- Validation Accuracy: 95.182%
- False Rejection Rate for ‘apricot’: 82/1506
- False Positive Rate for ‘apricot’: 180/1604
- False Rejection Rate for ‘Marius’: 50/1459
- False Positive Rate for ‘Marius’: 86/1495
- Parameters: 23,332
- Input Tensor Shape: (1, 24, 32, 1)
- Sampling Rate: 1.2 samples/s

Model 2:

- Training Accuracy: 88.02%
- Validation Accuracy: 95.182%
- False Rejection Rate for ‘apricot’: 15/2978
- False Positive Rate for ‘apricot’: 63/3026
- False Rejection Rate for ‘Marius’: 760/3040
- False Positive Rate for ‘Marius’: 0/2280
- Parameters: 155,886
- Input Tensor Shape: (1, 24, 32, 1)
- Sampling Rate: 0.67 samples/s

Model 3:

- Training Accuracy: 93.20%
- Validation Accuracy: 96.91%
- False Rejection Rate for ‘apricot’: 6/2978
- False Positive Rate for ‘apricot’: 116/3088
- False Rejection Rate for ‘Marius’: 38/3040
- False Positive Rate for ‘Marius’: 158/3160
- Parameters: 23,332
- Input Tensor Shape: (1, 24, 32, 1)
- Sampling Rate: 1.2 samples/s

Discussion:

Overall, the team believes the third model (the one that is being submitted) performed very well. It is somewhat usable in its current form, but it is not without flaws. Currently, the team believes the best way to improve the model would be to identify what is causing the "apricot" keyword to constantly be recognized and to reduce the detection time. The majority of the difficulties that occurred during this project were not anything conceptually related; instead, the difficulties stemmed from bugs and the team's inexperience with working with Arduino products.

Conclusion:

In conclusion, the team set out to create a model which could successfully identify two separate keywords. After various methods of data collection and multiple models were assembled, the team was able to successfully create and deploy a trained model which could correctly identify if a keyword was spoken. The final model was faster and more accurate than the team's first attempt. The team learned better techniques to construct datasets and how to analyze, diagnose, and correct flaws in pre-existing models.

References:

https://github.com/ArduCAM/pico-tflmicro