Assignment5: TinyML

Submitted by: Soham Lakhote

Q.1.1) Does the model perform as accurately as expected on your smartphone? List a few methods to improve the model's accuracy.

Answer:
No, the model does not always perform as accurately as expected. At times, it produces false positives by classifying unrelated words under the "hello world" label.

This discrepancy can occur because microphones in edge devices, such as smartphones, may have different sensitivities or frequency responses compared to the audio data used during model training. As a result, the model may misinterpret audio signals.

To address this issue:

1. **Fine-tune the model** using data recorded from the target hardware, such as the smartphone's microphone, and include this data in the training process.
2. The original training dataset did not include my own voice samples, which could cause the model to misclassify words spoken in my voice. During testing on the laptop, I used a dataset that did not contain my voice, but for live classification on the mobile phone, I am using my voice. This creates an **"apples-to-oranges" comparison** between the two scenarios.
3. To improve accuracy, I should include **labeled voice data of my own** in the training dataset to help the model adapt to my voice characteristics.

Q.1.2) When building a model for resource-limited hardware, how do you balance fast inference times with acceptable model accuracy? What trade-offs did you encounter?

Answer:

Strategies for Balancing Inference Speed and Accuracy:

1. **Model Simplification:**
   - Use smaller model architectures (e.g., MobileNet, TinyML models).
   - Quantize the model (e.g., convert weights and activations from 32-bit floats to 8-bit integers).
   - Prune unimportant connections or neurons to reduce model size and complexity and retrain it. Do the sensitivity analysis while pruning w.r.t accuracy/energy efficiency for deciding which nets to prune.
2. **Data Preprocessing:**
   - Use efficient feature extraction methods to reduce input size without losing critical information (e.g., downsample audio or resize images).
   - Perform dimensionality reduction techniques on input data.

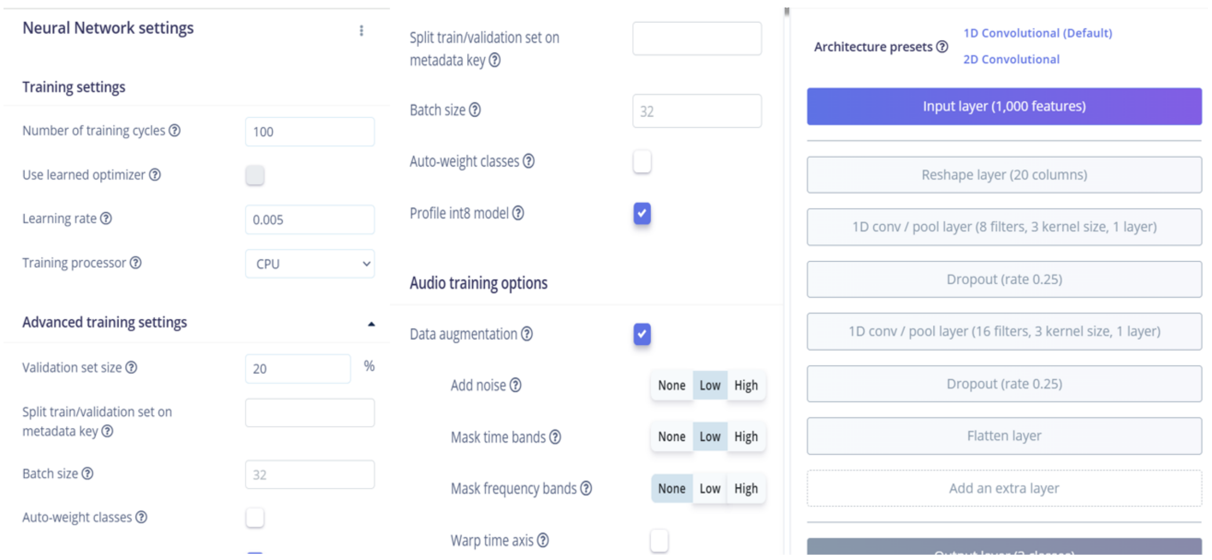
3. **Hardware-Aware Optimization:**
   - Design models specifically optimized for the target hardware (e.g., EdgeTPU, ARM Cortex-M processors).
   - Use compiler-level optimizations (e.g., TensorFlow Lite, Edge Impulse's deployment tools).
4. **Fine-Tuning:**
   - Employ transfer learning or fine-tune pre-trained models to achieve acceptable accuracy while starting with an efficient base model.
   - Train with hardware-specific data for better real-world performance.
5. **Batch Processing:**
   - Use smaller batch sizes or even single-input inference, as most edge devices cannot handle large parallel computations due to limited computation power.

Common Trade-Offs Encountered:

1. **Accuracy vs. Model Size:**
   - Reducing model size to fit into limited memory (e.g., by pruning or quantization) often leads to slight degradation in accuracy.
   - Complex models with high accuracy may exceed hardware memory or processing constraints.
2. **Inference Speed vs. Complexity:**
   - Achieving faster inference often requires simpler architectures, which might compromise the model's ability to capture intricate patterns in data.
   - Complex models yield better accuracy but lead to higher latency and energy consumption.
3. **Energy Efficiency vs. Model Depth:**
   - Shallow models consume less power but might lack the depth required for nuanced decisions.
   - Deeper models perform better but are less suitable for energy-constrained environments.
4. **Generalization vs. Hardware-Specific Optimization:**
   - Models trained and optimized for specific hardware might lose generalization capabilities on other devices or datasets.
5. **Data Quality vs. Compute Constraints:**
   - Using high-quality, detailed input data improves accuracy but can be computationally expensive. For example, reducing image resolution or audio sampling rates saves resources but risks losing important details.

Screenshots of the Deployment Process:

NN Classifier Settings:



Neural Network settings

Training settings

Number of training cycles	100
Use learned optimizer	☐
Learning rate	0.005
Training processor	CPU

Advanced training settings

Validation set size	20 %
Split train/validation set on metadata key	
Batch size	32
Auto-weight classes	☐
Profile int8 model	✓

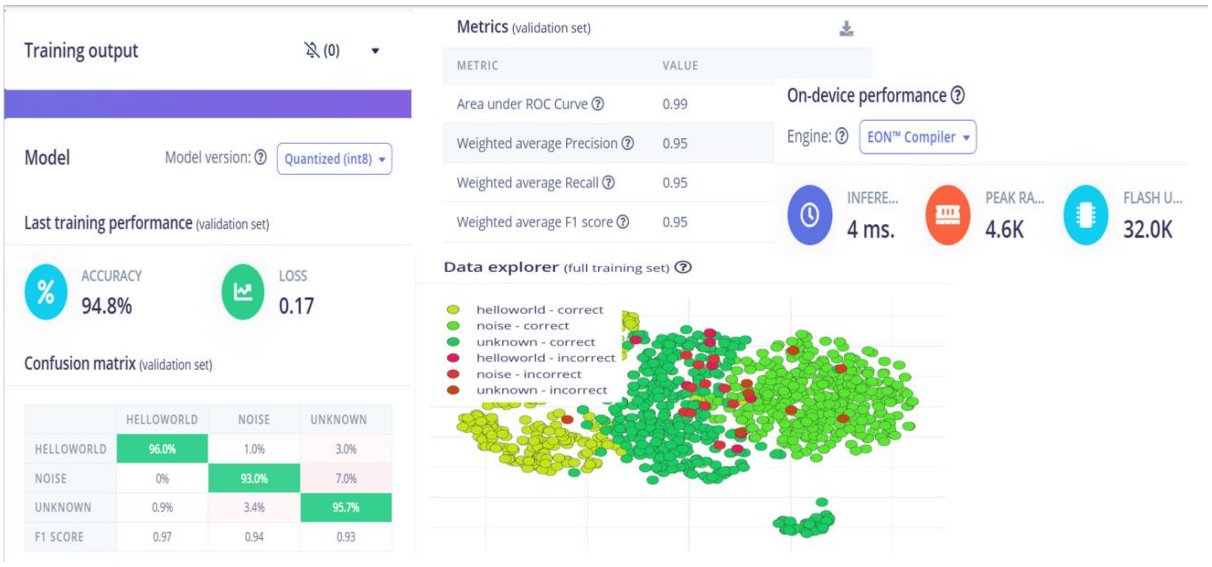
Audio training options

Data augmentation	✓
Add noise	None [✓ Low] High
Mask time bands	None [✓ Low] High
Mask frequency bands	[✓ None] Low High
Warp time axis	☐

Architecture presets: 1D Convolutional (Default) | 2D Convolutional

- Input layer (1,000 features)
- Reshape layer (20 columns)
- 1D conv / pool layer (8 filters, 3 kernel size, 1 layer)
- Dropout (rate 0.25)
- 1D conv / pool layer (16 filters, 3 kernel size, 1 layer)
- Dropout (rate 0.25)
- Flatten layer
- Add an extra layer

Training Output Results: (Few of the misclassified data during testing was incorporated back into the training and that's why the accuracy is higher).



Training output

Model — Model version: Quantized (int8)

Last training performance (validation set)

ACCURACY 94.8% — LOSS 0.17

Confusion matrix (validation set)

	HELLOWORLD	NOISE	UNKNOWN
HELLOWORLD	96.0%	1.0%	3.0%
NOISE	0%	93.0%	7.0%
UNKNOWN	0.9%	3.4%	95.7%
F1 SCORE	0.97	0.94	0.93

Metrics (validation set)

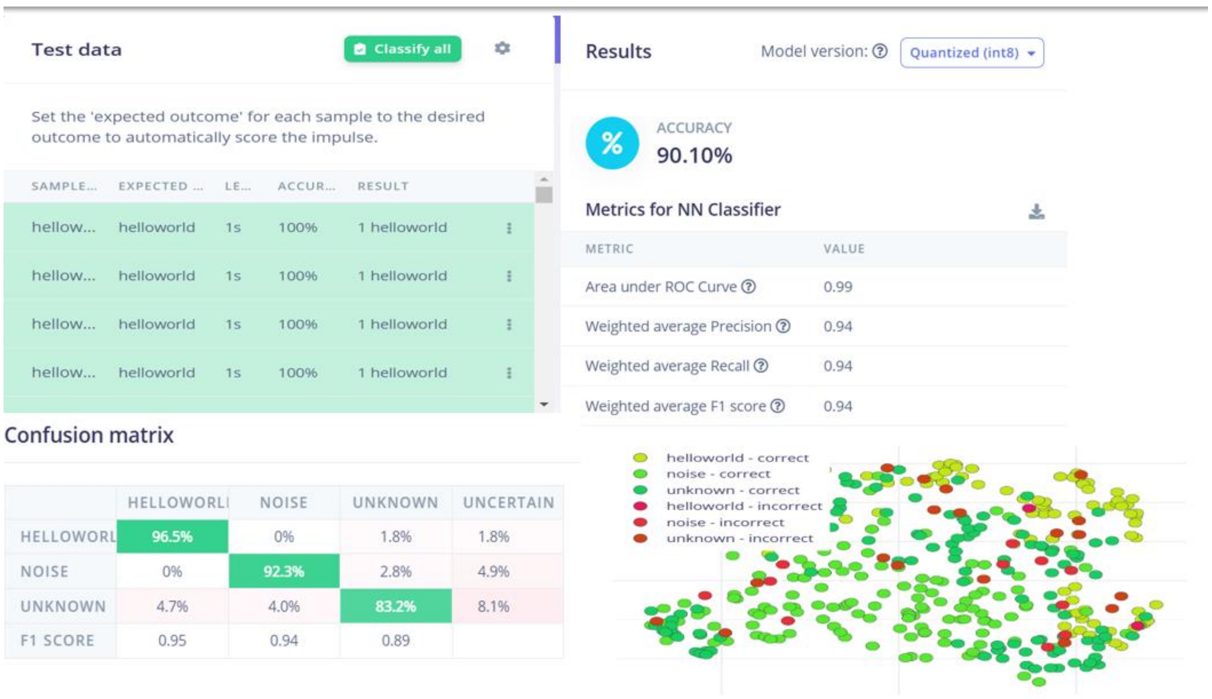
METRIC	VALUE
Area under ROC Curve	0.99
Weighted average Precision	0.95
Weighted average Recall	0.95
Weighted average F1 score	0.95

On-device performance — Engine: EON™ Compiler

INFERE... 4 ms. — PEAK RA... 4.6K — FLASH U... 32.0K

Data explorer (full training set)

<u>Testing Output Results:</u>


Test data

Classify all

Set the 'expected outcome' for each sample to the desired outcome to automatically score the impulse.

SAMPLE...	EXPECTED ...	LE...	ACCUR...	RESULT
hellow...	helloworld	1s	100%	1 helloworld
hellow...	helloworld	1s	100%	1 helloworld
hellow...	helloworld	1s	100%	1 helloworld
hellow...	helloworld	1s	100%	1 helloworld

Results

Model version: Quantized (int8)

ACCURACY
90.10%

Metrics for NN Classifier

METRIC	VALUE
Area under ROC Curve	0.99
Weighted average Precision	0.94
Weighted average Recall	0.94
Weighted average F1 score	0.94

Confusion matrix

	HELLOWORLI	NOISE	UNKNOWN	UNCERTAIN
HELLOWORL	96.5%	0%	1.8%	1.8%
NOISE	0%	92.3%	2.8%	4.9%
UNKNOWN	4.7%	4.0%	83.2%	8.1%
F1 SCORE	0.95	0.94	0.89	

Q.2) Record and provide links to:

The keyword-spotting model working on your smartphone:

Video Link: [Edge Impulse Model on Mobile Phone.mp4]

The keyword-spotting model working on the embedded Arduino board.

Video Link: [Edge Impulse Model on Arduino.mp4]

Q.3) **Reflections**:

Share your experience deploying the model to your smartphone and Arduino board. Mention any technical difficulties or interesting observations.

<u>Answer:</u>

Observations and Insights from Deployment

The deployment experience was insightful, revealing several areas for improvement in the model's performance. The following are the key observations:

1. Observations on Arduino:

The model classified the following words as "Hello World," indicating false positives:

- **Hello**
- **World**
- **Crocodile**
- **Mouse**

2. Observations on Mobile:

When deployed on a smartphone, the model classified these words as "Hello World":

- **Dog**
- **Mouse**
- **Buffalo**

3. General Observations:

- **Environment-Specific Behavior:**
  While testing in the lab, the smartphone-deployed model exhibited a significant number of false positives. However, when tested in a quiet room, the false positives were notably reduced. This highlights the impact of environmental noise on model performance and emphasizes the need for training the model with data collected directly from the target hardware's sensor under varied acoustic conditions.
- **Lack of Voice Representation:**
  The model's sensitivity to false positives appears to stem, in part, from the underrepresentation of Indian accents and voices in the training dataset. As a result, it misclassifies other words with similar phonetic structures.
- **Plan for Improvement:**
  To address these issues:
  1. **Hardware-Specific Data Collection:** Collect data directly from the target hardware, such as Arduino or the smartphone, under different environmental conditions. Properly label this data and incorporate it into the training process.
  2. **Inclusion of Diverse Voice Samples:** Add voice samples, including my own, to the training dataset to improve the model's accuracy with Indian accents and voices.
  3. **Retrain and Re-evaluate:** After augmenting the dataset, retrain the model and re-test it to measure improvements in classification accuracy.
