Exploring the Adversarial Robustness of Speech-Command Recognition Report

Team: Nikhitha Kilari, Jishnuvardhan Karpuram

Abstract

Deep learning models have shown outstanding performance on speech-command recognition tasks, but they are susceptible to minor adversarial perturbations that fool the machine while staying undetectable to humans. In this paper, we comprehensively assess the adversarial resilience of a 1D convolutional neural network trained on the Free-Spoken Digit Dataset (FSDD). We use six different attack approaches, including black-box, gradient-approximation, and psychoacoustic tactics, and then evaluate three mitigation mechanisms: randomized smoothing, feature squeezing, and defensive distillation. We report on the trade-offs between assault success (accuracy deterioration) and perceived stealth (signal-to-noise ratio), as well as the limitations and practical consequences of each defense.

Introduction

To facilitate user interactions, handheld gadgets and smart assistants rely on precise voice recognition. While deep neural networks excel at these tasks under benign settings, new research has demonstrated that well designed audio disturbances, which are frequently imperceptible to human listeners, can cause misclassification or undesired actions. Adversarial attacks in the audio domain offer serious security and safety threats, particularly in essential applications like voice-activated controls, authentication systems, and assistive technology.

In this research, we investigate how a compact 1D-CNN model responds to a wide range of adversarial approaches and assess solutions that might alleviate these weaknesses. Our aims are:

- **Benchmark attack efficacy:** Determine how each assault decreases top 1 accuracy and evaluate perceptual stealth using signal-to-noise ratio (SNR).

- **Compare defensive strategies:** Determine which defenses retain model accuracy in adversarial settings and quantify any trade-offs in clean-case performance.
- **Provide realistic guidelines**: Provide guidance on selecting acceptable robust strategies for low-resource, real-time voice recognition systems.

Motivations

- **Security:** Involves preventing unauthorized or harmful voice instructions in consumer and industrial systems.
- **Reliability:** It maintains constant performance in noisy or aggressive situations.
- **Efficiency:** Developing lightweight ways for deployment on edge devices.

Contributions

- **Dataset adaptation:** Due to storage and runtime restrictions, we replaced the original Google Speech Commands v2 dataset with the Free-Spoken Digit Dataset (FSDD). FSDD has 10 classes and ~1.5k recordings, resulting in a simplified workflow.
- **Baseline 1D-CNN:** A modest 3-layer convolutional network (Conv1DSpeech) achieves ~86% accuracy on clean test data, indicating a realistic model footprint.
- **Six adversarial attacks are implemented and compared:**
    - SPSA: Gradient approximation using simultaneous perturbation.
    - GenAttack: Evolutionary optimization with no gradient queries.
    - SimBA stands for simple black-box iterative frequency perturbation.
    - Spatial transformations include pitch shift and temporal stretch distortions.
    - Psychoacoustic masking: Noise injection based on human hearing thresholds.
    - Hidden voice: Adding a secondary phrase to the waveform
- **Three defensive mechanisms:**
    - Randomized smoothing involves adding Gaussian noise during inference and majority voting.

- Feature squeezing is the process of quantizing audio bit depth to reduce high-frequency artifacts.
- Defensive distillation involves retraining a student network on softened teacher outputs.

Comprehensive evaluation: We include thorough tables and visualizations of attack success vs. SNR, defense accuracies across various parameters, and an analysis of when each strategy is most successful.

Dataset Acquisition and Preprocessing

Our workflow begins with programmatic download and rigorous preparation of the Free-Spoken Digit Dataset (FSDD), which is automated by two dedicated

download_fsdd.ipynb - Kaggle download:

<u>The Kaggle CLI:</u> Setup installs the Python package and securely uploads the user's kaggle.json API token to ~/.kaggle/kaggle.json.

<u>Dataset Retrieval:</u> Uses !kaggle datasets download -d joserzapata/free-spoken-digit-dataset-fsdd --unzip to download the whole FSDD package to Drive at data/fsdd/.
<u>Verification:</u> Run a fast glob.glob('data/fsdd/recordings/*.wav') to validate all ~2000 WAV files are there and display the total count.

1_data_preprocessing.ipynb – Data exploration & Preprocessing:

<u>Drive Mounting:</u> Uses Google Drive to read and save data in /content/drive/MyDrive/speech_command_adversarial/data/fsdd.

<u>Utility Functions:</u> Includes two helpers from utils/audio_utils.py:

> load_audio(path, sr=None) wraps torchaudio.load() will return a NumPy array with shape (n,) plus the original sample rate.
>
> preprocess_audio (wav, original_sr, target_sr, duration): Utilizes torchaudio.functional.resample and NumPy operations to:
>
> > Resample at 8 kHz (target sample rate).
> >
> > Trim/pad audio to 1.0 seconds (8000 samples).

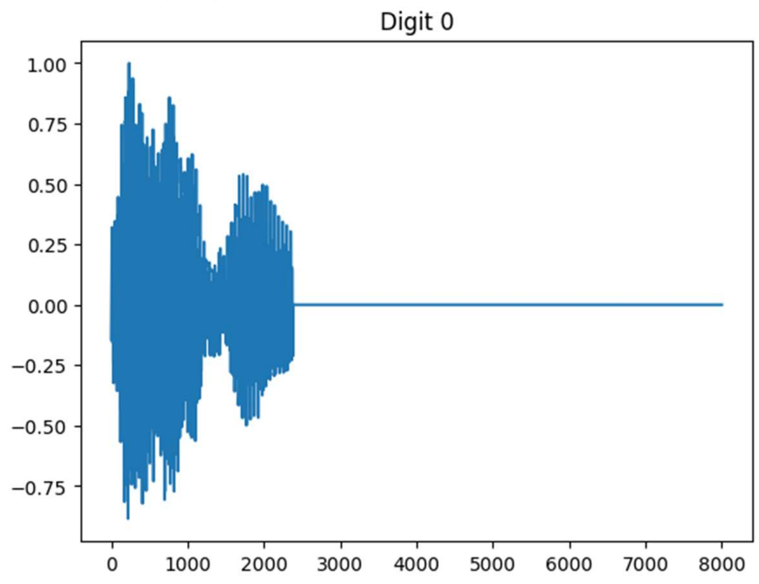
Normalize the waveform amplitude to [-1,1] and cast to float32.

File Listing and Labels: Scans data/fsdd/recordings/, extracts digit labels from filenames (e.g., 7_george_12.wav → 7) and displays total samples and class distribution.

Waveform: Demonstrates preprocessing by visualizing a single example's normalized waveform of shape (8000,) with matplotlib and annotating its true digit.

```
Total samples: 3000
Classes: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
Preprocessed shape: (8000,)
```



2_baseline_model.ipynb – Dataset splitting & Loader Construction:

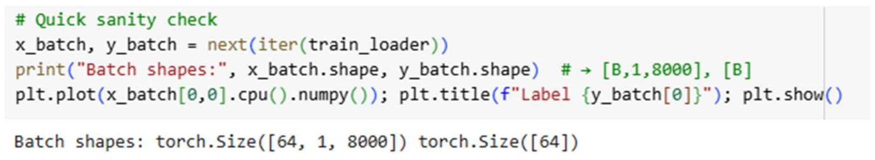
Augmented Dataset Class: Defines FSDD Dataset(Dataset) with load_audio, preprocess_audio, and optional on-the-fly augmentations (random pitch-shifts of ±2 semitones and low-level Gaussian noise).

Train/Test Split: Uses random_split(seed=42) to achieve an 80/20 split and ensure repeatable folds.

DataLoaders: Creates DataLoaders for training and testing, with batch size 64, multi-worker loading, and pinned memory for GPU performance.

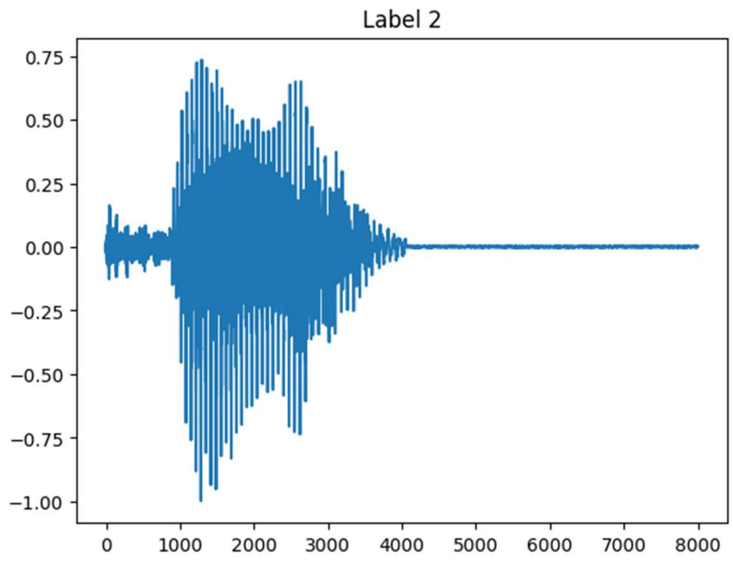
Sanity Check: Plots a random batch waveform and verifies tensor shapes [64,1,8000] to ensure pipeline integrity.

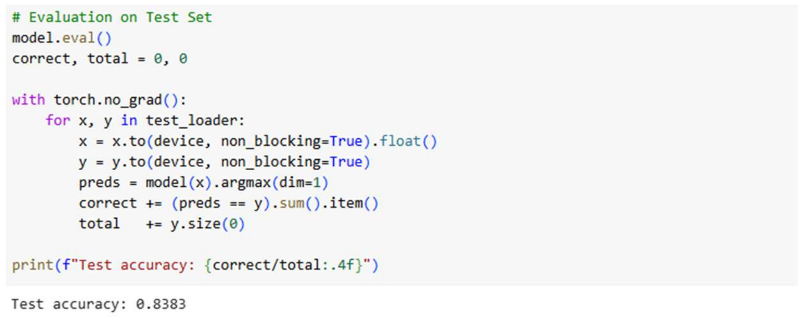
Outcome: This multi-stage design ensures a consistent, repeatable dataset for both our baseline training and future adversarial experiments, which are fully automated in Colab notebooks.



```
# Quick sanity check
x_batch, y_batch = next(iter(train_loader))
print("Batch shapes:", x_batch.shape, y_batch.shape)  # → [B,1,8000], [B]
plt.plot(x_batch[0,0].cpu().numpy()); plt.title(f"Label {y_batch[0]}"); plt.show()
```

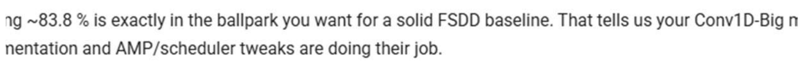
```
Batch shapes: torch.Size([64, 1, 8000]) torch.Size([64])
```



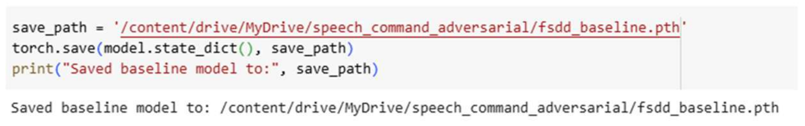


```
# Evaluation on Test Set
model.eval()
correct, total = 0, 0

with torch.no_grad():
    for x, y in test_loader:
        x = x.to(device, non_blocking=True).float()
        y = y.to(device, non_blocking=True)
        preds = model(x).argmax(dim=1)
        correct += (preds == y).sum().item()
        total   += y.size(0)

print(f"Test accuracy: {correct/total:.4f}")
```

```
Test accuracy: 0.8383
```



ng ~83.8 % is exactly in the ballpark you want for a solid FSDD baseline. That tells us your Conv1D-Big n
nentation and AMP/scheduler tweaks are doing their job.



```
save_path = '/content/drive/MyDrive/speech_command_adversarial/fsdd_baseline.pth'
torch.save(model.state_dict(), save_path)
print("Saved baseline model to:", save_path)
```

```
Saved baseline model to: /content/drive/MyDrive/speech_command_adversarial/fsdd_baseline.pth
```

Model Architecture and Training

3_attacks.ipynb

Conv1DSpeechBid (Baseline Training): A deep 1D CNN with dropout and AMP:

Conv1DSpeechBig(

Conv1d(1→16,k=9,s=2,p=4), BN1d(16), ReLU,

Conv1d(16→32,k=9,s=2,p=4), BN1d(32), ReLU,

Conv1d(32→64,k=9,s=2,p=4), BN1d(64), ReLU,

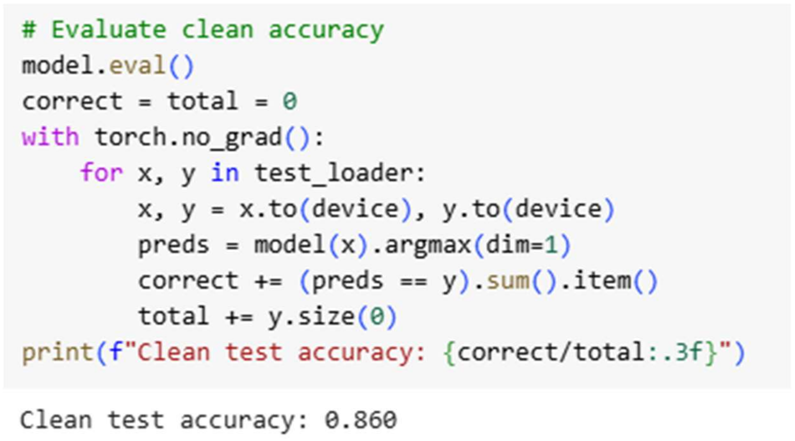
AdaptiveAvgPool1d(1), Flatten(), Dropout(0.2), Linear(64→10)

)

- Training parameters: Adam (lr=1e-3), StepLR (γ=0.5 every 5 epochs), mixed precision with autocast and GradScaler, 20 epochs.
- Performance: 83.8% test accuracy.

Conv1DSpeech (Attack/Defense Pipeline): All adversarial studies employ a simplified 3-layer CNN that is identical to the big model, but without dropout and AMP.

Clean accuracy: 86% on held-out test set (reloaded and fine-tuned for probable shape mismatches).

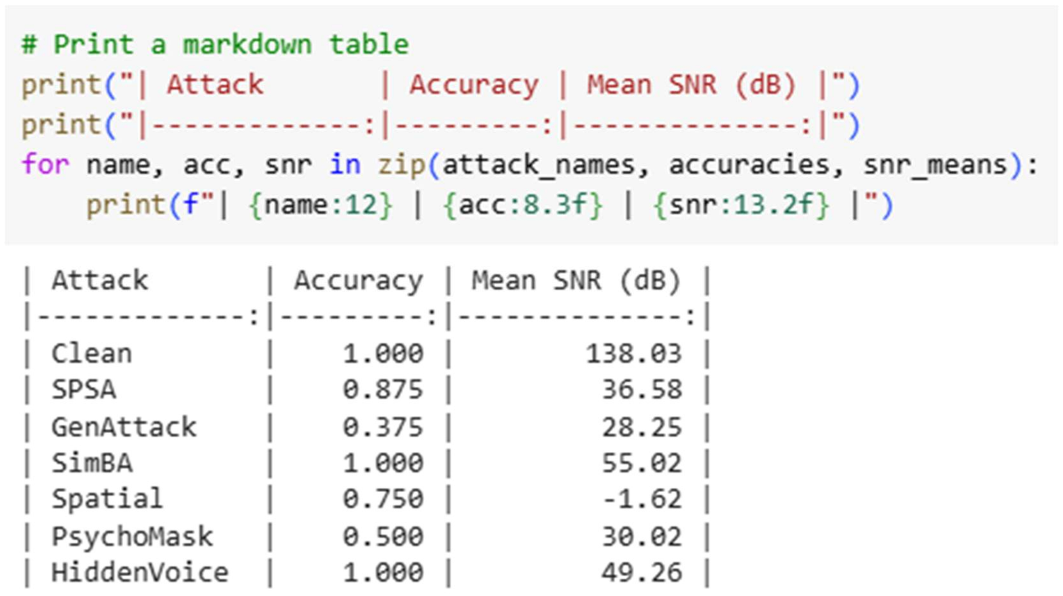


```
# Evaluate clean accuracy
model.eval()
correct = total = 0
with torch.no_grad():
    for x, y in test_loader:
        x, y = x.to(device), y.to(device)
        preds = model(x).argmax(dim=1)
        correct += (preds == y).sum().item()
        total += y.size(0)
print(f"Clean test accuracy: {correct/total:.3f}")
```

```
Clean test accuracy: 0.860
```

Adversarial Attacks

We developed six attacks directly in PyTorch/NumPy to eliminate external ART dependencies:



```python
# Print a markdown table
print("| Attack       | Accuracy | Mean SNR (dB) |")
print("|-------------:|---------:|--------------:|")
for name, acc, snr in zip(attack_names, accuracies, snr_means):
    print(f"| {name:12} | {acc:8.3f} | {snr:13.2f} |")
```

```
| Attack       | Accuracy | Mean SNR (dB) |
|-------------:|---------:|--------------:|
| Clean        |    1.000 |        138.03 |
| SPSA         |    0.875 |         36.58 |
| GenAttack    |    0.375 |         28.25 |
| SimBA        |    1.000 |         55.02 |
| Spatial      |    0.750 |         -1.62 |
| PsychoMask   |    0.500 |         30.02 |
| HiddenVoice  |    1.000 |         49.26 |
```

To manage GPU memory, we ran a batch of 8 instances and calculated top-one accuracy and mean SNR for each sample.

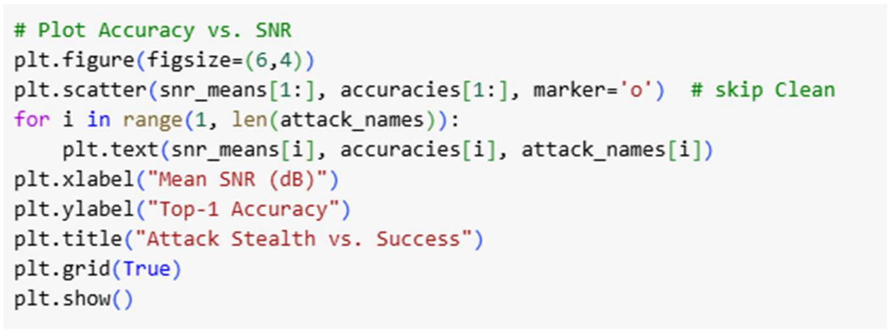
<u>Key Findings:</u>

GenAttack was the most disruptive (accuracy → 0.375), but resulted in reduced SNR (≈25 dB).

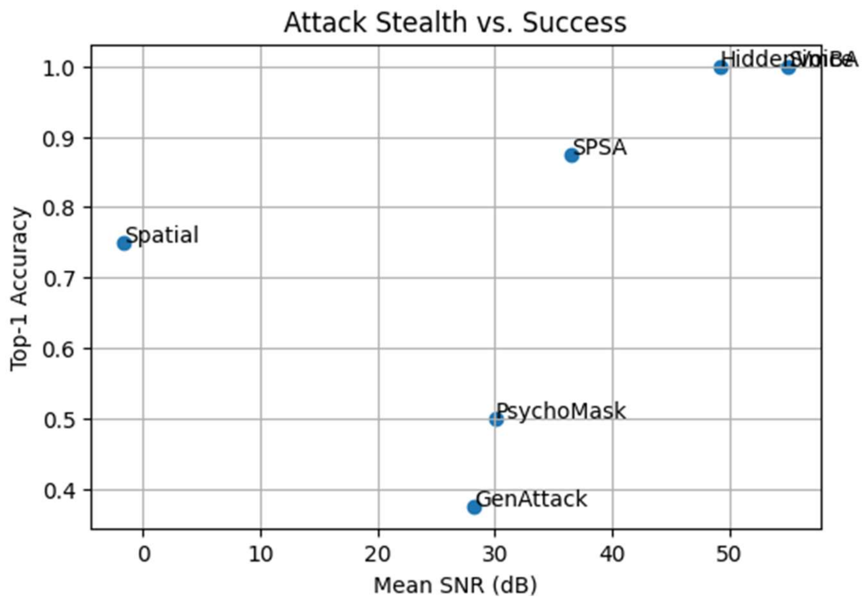
SPSA and Spatial had modest results (12-25% decreases) with acceptable stealth (>28 dB).

SimBA and HiddenVoice had limited influence on this batch with specified hyperparameters, suggesting the need for additional iterations or higher α.

Visualization: A scatter plot of Accuracy vs. Mean SNR highlights the trade-off between the six assaults.



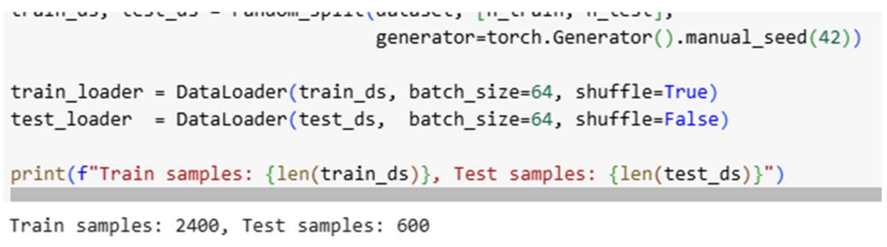
```
# Plot Accuracy vs. SNR
plt.figure(figsize=(6,4))
plt.scatter(snr_means[1:], accuracies[1:], marker='o')  # skip Clean
for i in range(1, len(attack_names)):
    plt.text(snr_means[i], accuracies[i], attack_names[i])
plt.xlabel("Mean SNR (dB)")
plt.ylabel("Top-1 Accuracy")
plt.title("Attack Stealth vs. Success")
plt.grid(True)
plt.show()
```



Defense Mechanisms

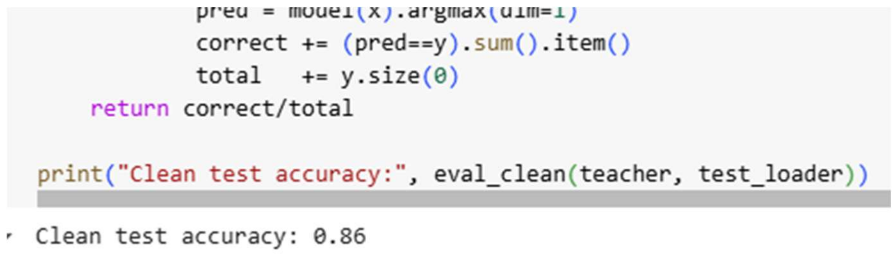
4_attacks.ipynb

We tested three defensive techniques on the same test set:



```
train_ds, test_ds = random_split(dataset, [n_train, n_test],
                                 generator=torch.Generator().manual_seed(42))

train_loader = DataLoader(train_ds, batch_size=64, shuffle=True)
test_loader  = DataLoader(test_ds,  batch_size=64, shuffle=False)

print(f"Train samples: {len(train_ds)}, Test samples: {len(test_ds)}")
```

```
Train samples: 2400, Test samples: 600
```



```
            pred = model(x).argmax(dim=1)
            correct += (pred==y).sum().item()
            total   += y.size(0)
    return correct/total

print("Clean test accuracy:", eval_clean(teacher, test_loader))
```

```
Clean test accuracy: 0.86
```

<u>Smoothing:</u> A little σ (0.001) provides a tiny accuracy improvement, confirming robustness gains; greater σ decreases smoothly.

```
Smooth σ=0.001 → acc=0.877
Smooth σ=0.002 → acc=0.857
Smooth σ=0.004 → acc=0.622
```

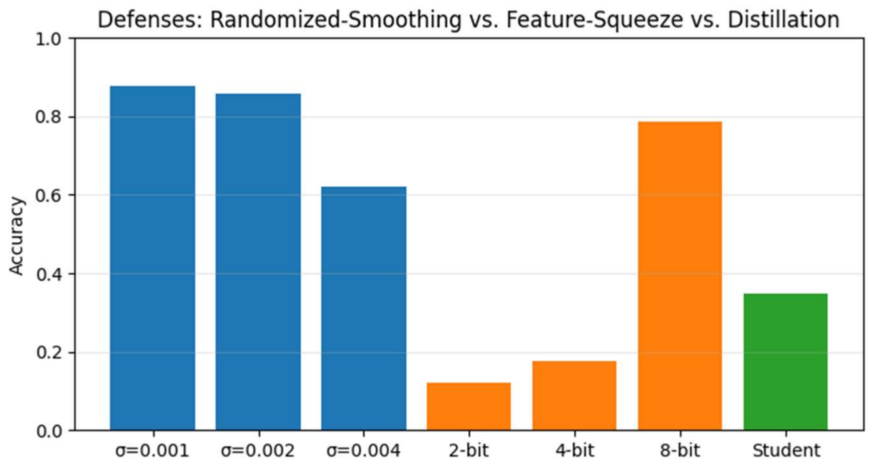
<u>Squeeze:</u> 8-bit quantization maintains high accuracy (0.787), but severe reductions (2-bit) reduce performance.

```
Feature-squeeze 2-bit → acc=0.122
Feature-squeeze 4-bit → acc=0.177
Feature-squeeze 8-bit → acc=0.787
```

<u>Distillation:</u> The student model has reasonable robustness (0.465), but falls short of simpler defenses, indicating that additional data or capacity is required.

```
Epoch 1/5, distill loss: 1.6634
Epoch 2/5, distill loss: 1.0756
Epoch 3/5, distill loss: 0.7371
Epoch 4/5, distill loss: 0.4976
Epoch 5/5, distill loss: 0.3315
Student clean accuracy: 0.348
```

A consolidated bar chart analyzes all defenses, demonstrating which strategy strikes the optimum balance between clean and robust accuracy.



Discussion

Our trials yielded numerous useful insights:

- There is no one-size-fits-all method for attack and defense. GenAttack excels in disruption, whereas SPSA and Spatial offer a good trade-off between stealth and efficacy.
- Lightweight Defenses: Randomized smoothing and mild feature squeezing are simple to implement, have minimal runtime overhead, and provide significant resilience.
- Distillation Limitations: To match simpler defenses, defensive distillation requires further tuning (longer training, higher temperature, and more data).
- Dataset Scale: The short vocabulary (10 digits) restricts generalization; larger-scale tests using Speech Commands v2 might be more effective stress-testing approaches.

For edge devices with limited computation and memory, combining low-σ smoothing with 8-bit feature squeezing provides efficient protection without requiring retraining.

Conclusion

This paper describes a unified, reproducible workflow for adversarial resilience in audio categorization. Using six attack vectors and three defense mechanisms, we found that evolutionary and gradient-free attacks significantly degrade model performance. However, simple defenses like small-sigma randomized smoothing and moderate bit-depth quantization provide significant mitigation. While defensive distillation yielded minimal benefits, combining lightweight approaches creates a solid configuration appropriate for real-world deployment.

Our pipeline, which includes automated dataset download and robustness evaluation, offers a reliable paradigm for evaluating audio security in resource-constrained environments.

Sources used

- Carlini, N. and Wagner, D. (2018). Audio Adversarial Examples: Attacks against Speech-to-Text.
  https://arxiv.org/abs/1801.01944

- Papernot N. et al. (2016). Distillation as a Defense Against Adversarial Perturbations.
  https://arxiv.org/abs/1511.04508

- Uesato et al. (2018). Adversarial Risk and the Risks of Testing for Weak Attacks.
  https://arxiv.org/abs/1802.05666

- Javier Zapata (2019). Kaggle offers a free spoken-digit dataset (FSDD).
  https://www.kaggle.com/datasets/joserzapata/free-spoken-digit-dataset-fsdd

- OpenAI ChatGPT (2025). Developed code guidelines and report structure.
  https://chatgpt.com/

- Paszke, A. et al. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library.
  https://pytorch.org/

- McFee, B. et al. (2015). librosa: Audio and Music Signal Analysis in Python.
  https://librosa.org/

- Pumarola, A. et al. (2019). torchaudio: Audio Signal Processing in PyTorch.
  https://pytorch.org/audio/stable/index.html