CIS 602 Fundamentals of Deep Learning
Fall 2025

Project 3
GRU & LSTM RNNs for Audio Recognition
Due: November 26, 2025

Michael Evan 02081213
Graduate Computer Science Department
UMassDartmouth

Abstract

Train and compare two PyTorch RNN (recurrent neural network) types, GRU and LSTM to recognize 12 common spoken words. Process audio .wav clips, convert into mel-spectrogram features, keep individual speakers separated in one of train/val/test sets with training utilizing the Adam optimizer & Cross-Entropy Loss function. Compare accuracy and metrics of both the GRU and LSTM models with the use of various plots and statistical metrics. Utilize batch training and a lightweight real-time Python inference with a rolling audio buffer. Develop a project that is simple, reliable, and reproducible on a local machine.

Introduction

Relevance & Importance

Small-vocabulary speech recognition is common in voice UIs, accessibility, and IoT fast, reliable models matter. Comparing GRU vs LSTM shows tradeoffs in speed, accuracy, and deployment cost for edge/local systems. Speaker-disjoint splits and clear diagnostics improve reproducibility and fairness in evaluation.

Task definition

Input: 12 unique words (yes, no, bird, cat, dog, up, down, left, right, stop, go, forward) distributed across samples of individual 16kHz/16bit .wav audio files. These are converted to mel-spectrogram (frames=100, n_mels=40) -> sequence length 100, feature dim 40.

Output: accuracy of predictability of test data set of new unique speaker samples of the 12 unique word set, with optional Python inference for live spoken word sample accuracy of predictability.

Goal: Train RNN classifiers (GRU and LSTM) to recognize 12 spoken words from .wav clips. Use mel-spectrogram features, keep speakers disjoint across train/val/test, and support both batch training and live inference testing.

Major contributions

- **Data pipeline:** Speaker-aware splitting, per-class quotas, and labeled .TSV for easy dataset reuse.

- **Feature engineering:** On-the-fly mel-spectrogram flow. Adam Optimizer & Cross Entropy Loss function. Early stopping adjustable for epoch runs.

- **Model comparison:** Matched GRU and LSTM training with checkpointing and early stopping if desired; empirical accuracy and timing comparison.

- **Diagnostics & visualization:** Learning Curves of train/val loss, Precision/Recall/ f1/support diagnostics. Confusion Matrix & heat maps (normalized & counts). PCA of penultimate features and true-vs-pred plots to reveal class separability and confusions.

- **Bonus: Real-time demo script:** Lightweight rolling-buffer inference script for live microphone recognition with VAD (voice activity detection) & predictability.

Methodology

Data Set: The original dataset acquired was created by Pete Warden of Google Brain. The main dataset has a total of 35 words recorded by 2,618 individual speakers with each spoken word recorded at 16kHz sample rate - 16 bits/sample as a mono .wav file of ~1 second in length. There is a total of 105,829 utterances of the 35 words, each in their own directory, with a file size of 3.8GB compressed as a zip file of 2.7GB for download.

Data Prep: The data is available as a direct download from the link provided: http://download.tensorflow.org/dataspeech_commands_v0.02.tar.gz

After being extracted from the zip file, the data is stored in "**extracted_speech_commands**" directory which must be placed in the **.ipynb** working directory. The JupyterNotebook references the path to the data directory

"**SRC**", creates a new output directory "**my_out_dir**". Twelve of 35 words are referenced as **TARGETS** = ["yes", "no", "bird", "cat", "dog", "up", "down", "left", "right", "stop", "go", "forward"]. The words are extracted, speaker-unique split and placed in **train, test, val** directories which are created in **my_out_dir.** Each speaker has a unique 8 character hexadecimal ID appended to each file name for identification. A **clips.tsv** file is created in my_out_dir referencing all the .wav files to their respective train, val, test directories. Note, If my_out_dir already exists no new split is performed.

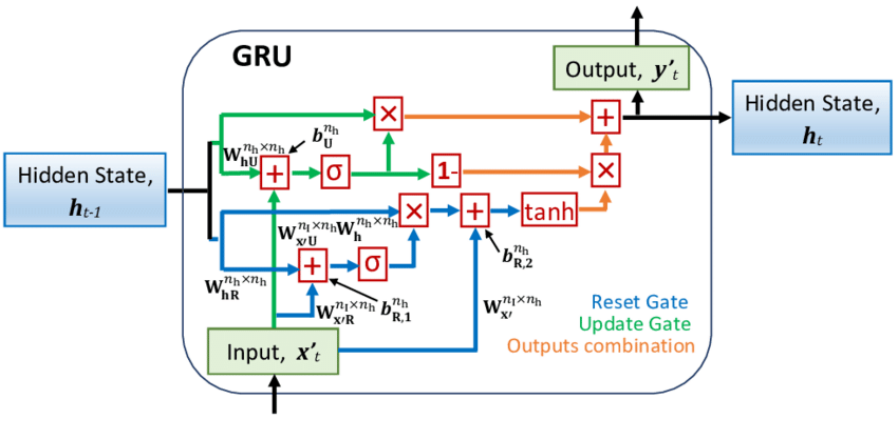
Prior to training, a mel-spectrogram converts audio waveforms into a time-frequency image using mel-scaled filterbanks, highlighting perceptually important speech patterns so models learn discriminative features faster and more robustly than from raw waveforms. The mel-spectrogram is stored in the form of a 2-dimensional array.

Neural Network Architecture: Two RNNs (GRU & LSTM) were chosen for the project for comparison.

GRU:

- Type: nn.GRU
- Layers: 2
- Hidden size: 128
- Bidirectional: No (used single-direction in code)
- Output head: Linear(128 -> num_classes) (uses last time-step)

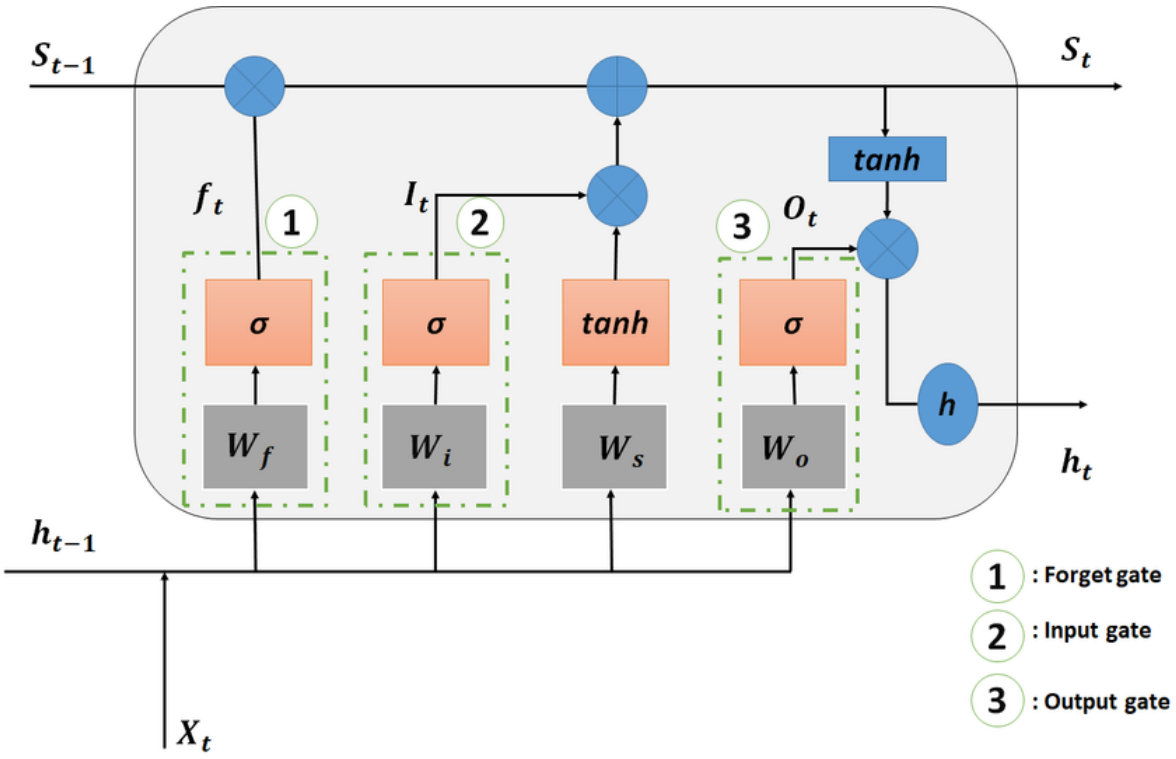
Fig 1): example GRU model:



LSTM:

- Type: nn.LSTM
- Layers: 2
- Hidden size: 128
- Output head: Linear(128 -> num_classes) (uses last time-step)

Fig 2): example LSTM model



Training

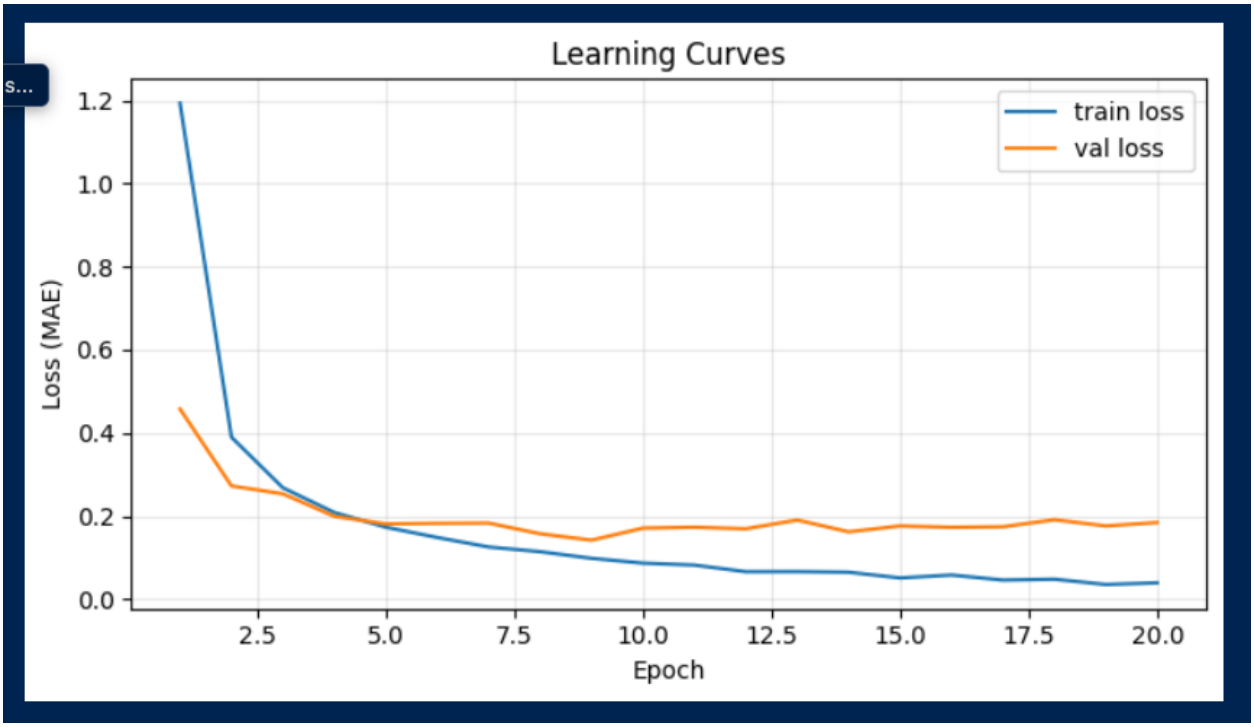
- Data prep: collect .wav files, speaker-unique 80/10/10 split, copy to respective output directories and write clips.tsv file.
- Features: convert each clip to mel-spectrogram on the fly.
- Dataloaders: build PyTorch Dataset/DataLoader (batch size 64).
- Model init: create GRU and LSTM models (same input dims, hidden size 128, 2 layers) and send to device (MPS/CPU/GPU).
- Optimizer & loss function: Adam optimizer (lr=1e-3) + CrossEntropyLoss.
- Weight decay: set in optimizer (Adam weight_decay)
- Gradient clipping: torch.nn.utils.clip_grad_norm_ to stabilize training.
- Training loop: train epochs (up to 20), run validation each epoch, compute val_acc, early stop with patience =10.
- Checkpointing: save best model file (best_gru.pt / best_lstm.pt) with model_state, optimizer_state, epoch, val_acc, and labels order.
- Data augment (specAugment, noise): helps speech models more than extra model regularizers.

Figures for Evaluation

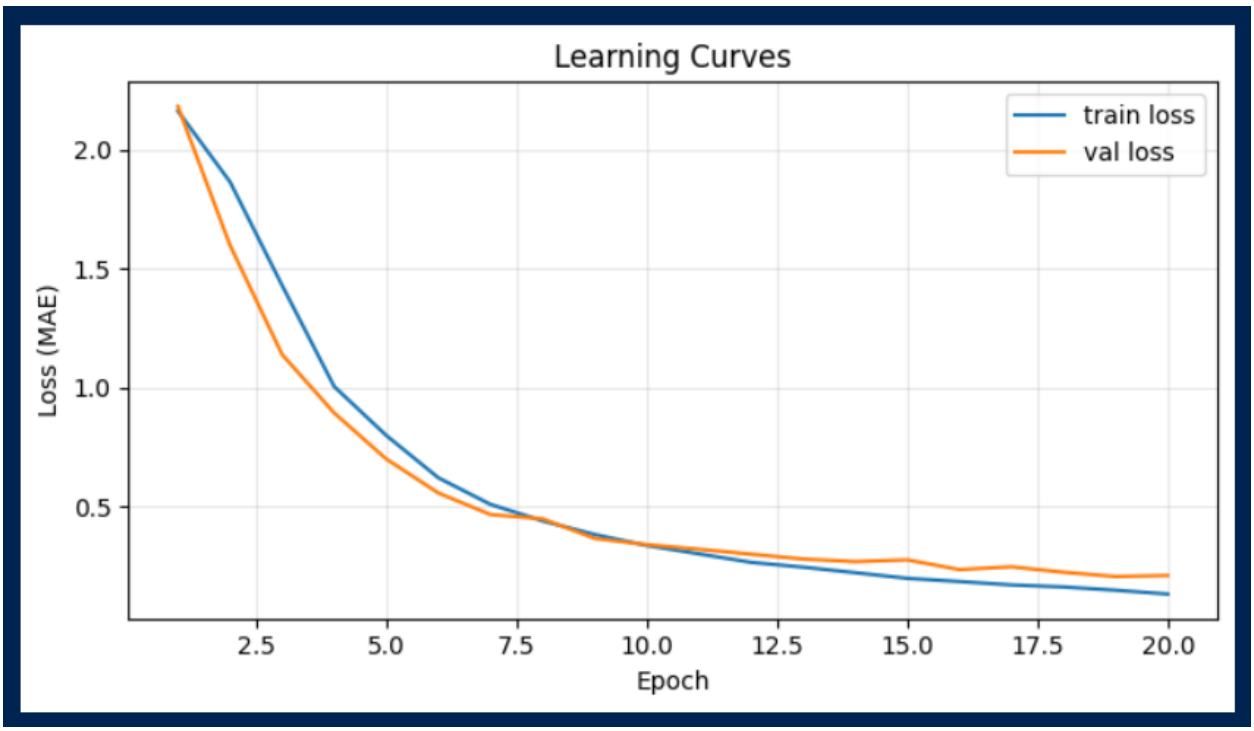
- Training curves (loss and validation accuracy vs epoch): Shows convergence, overfitting, and best-epoch selection.
- Confusion matrix heatmap (normalized / raw) Shows per-class confusion rates numerically and visually.
- Classification report table (precision / recall / F1 / support)
- Per-class numeric performance summary used alongside the confusion matrix.
- Per-class dataset counts / summary table: Reports quotas vs actuals (train/val/test) to verify class balance and splits.

Model Comparisons

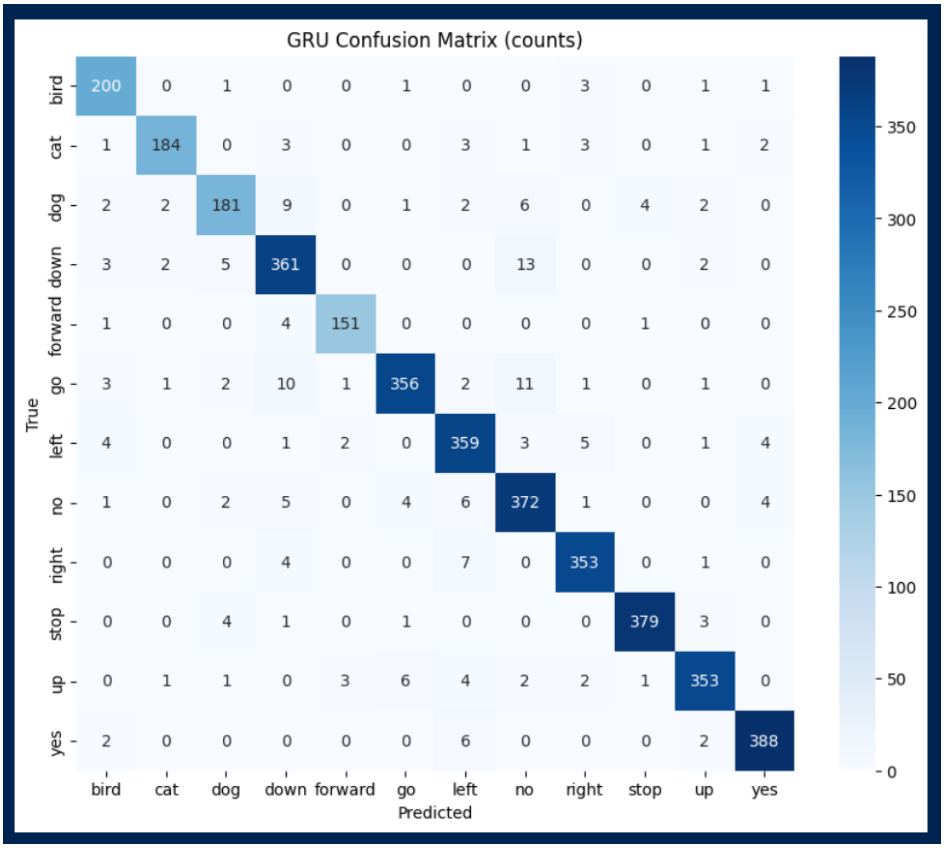
GRU best_gru.pt (epoch 17, val_acc = 95.60%)



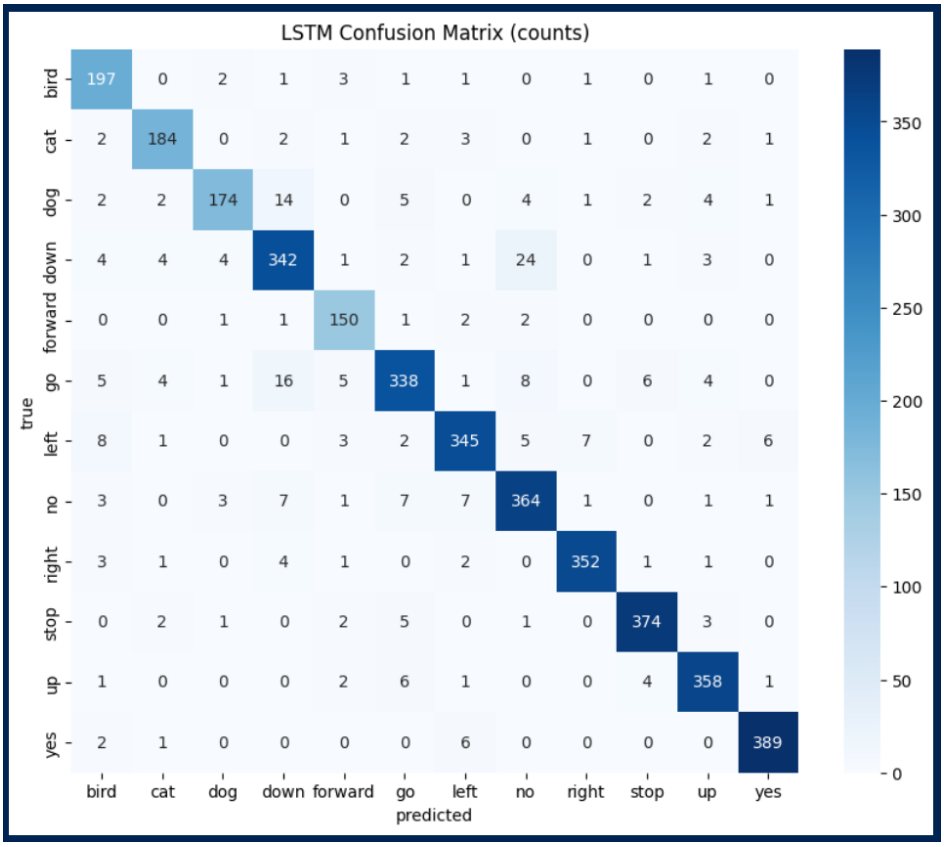
LSTM best_lstm.pt (epoch 20, val_acc = 93.70%)



GRU


GRU Confusion Matrix (counts)
True
Predicted
bird cat dog down forward go left no right stop up yes
bird 200 0 1 0 0 1 0 0 3 0 1 1
cat 1 184 0 3 0 0 3 1 3 0 1 2
dog 2 2 181 9 0 1 2 6 0 4 2 0
down 3 2 5 361 0 0 0 13 0 0 2 0
forward 1 0 0 4 151 0 0 0 0 1 0 0
go 3 1 2 10 1 356 2 11 1 0 1 0
left 4 0 0 1 2 0 359 3 5 0 1 4
no 1 0 2 5 0 4 6 372 1 0 0 4
right 0 0 0 4 0 0 7 0 353 0 1 0
stop 0 0 4 1 0 1 0 0 0 379 3 0
up 0 1 1 0 3 6 4 2 2 1 353 0
yes 2 0 0 0 0 0 6 0 0 0 2 388
350 300 250 200 150 100 50 0

LSTM


LSTM Confusion Matrix (counts)
true
predicted
bird cat dog down forward go left no right stop up yes
bird 197 0 2 1 3 1 1 0 1 0 1 0
cat 2 184 0 2 1 2 3 0 1 0 2 1
dog 2 2 174 14 0 5 0 4 1 2 4 1
down 4 4 4 342 1 2 1 24 0 1 3 0
forward 0 0 1 1 150 1 2 2 0 0 0 0
go 5 4 1 16 5 338 1 8 0 6 4 0
left 8 1 0 0 3 2 345 5 7 0 2 6
no 3 0 3 7 1 7 7 364 1 0 1 1
right 3 1 0 4 1 0 2 0 352 1 1 0
stop 0 2 1 0 2 5 0 1 0 374 3 0
up 1 0 0 0 2 6 1 0 0 4 358 1
yes 2 1 0 0 0 0 6 0 0 0 0 389
350 300 250 200 150 100 50 0

GRU

Overall test acc: 0.946396

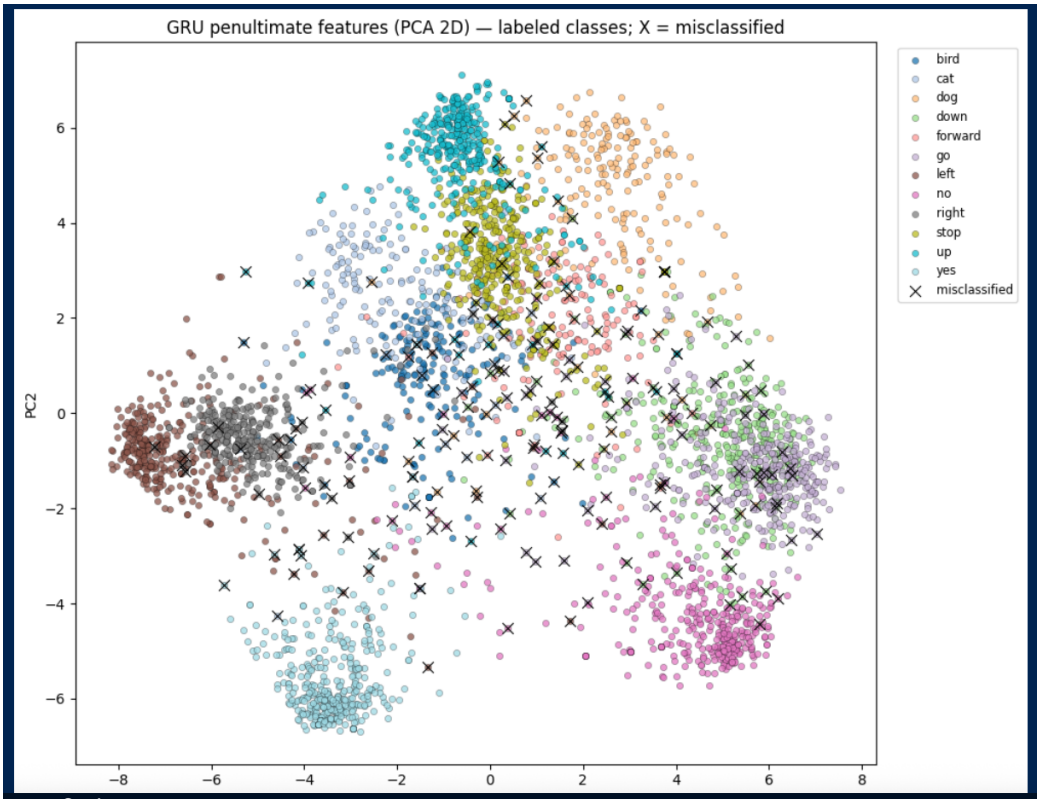
	precision	recall	f1-score	support
bird	0.92	0.97	0.94	207
cat	0.97	0.93	0.95	198
dog	0.92	0.87	0.89	209
down	0.91	0.94	0.92	386
forward	0.96	0.96	0.96	157
go	0.96	0.92	0.94	388
left	0.92	0.95	0.93	379
no	0.91	0.94	0.93	395
right	0.96	0.97	0.96	365
stop	0.98	0.98	0.98	388
up	0.96	0.95	0.95	373
yes	0.97	0.97	0.97	398
accuracy			0.95	3843
macro avg	0.95	0.94	0.95	3843
weighted avg	0.95	0.95	0.95	3843

LSTM

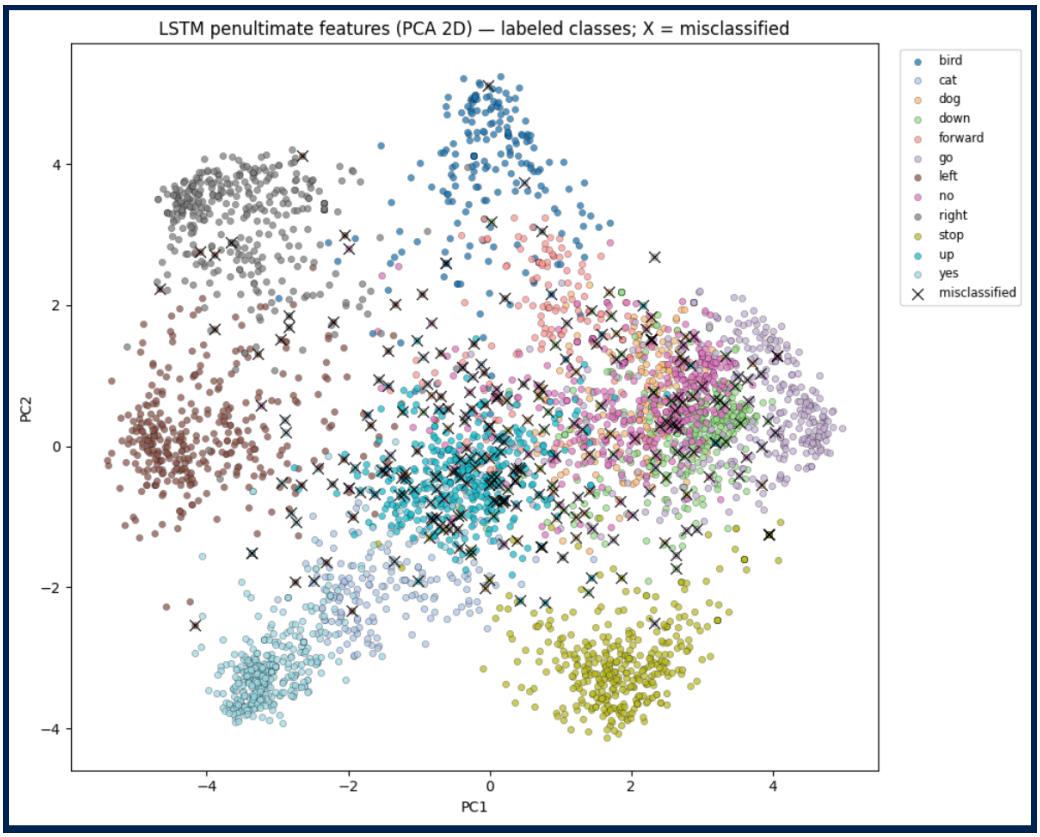
Overall test acc: 0.928181

	precision	recall	f1-score	support
bird	0.87	0.95	0.91	207
cat	0.92	0.93	0.93	198
dog	0.94	0.83	0.88	209
down	0.88	0.89	0.88	386
forward	0.89	0.96	0.92	157
go	0.92	0.87	0.89	388
left	0.93	0.91	0.92	379
no	0.89	0.92	0.91	395
right	0.97	0.96	0.97	365
stop	0.96	0.96	0.96	388
up	0.94	0.96	0.95	373
yes	0.97	0.98	0.98	398
accuracy			0.93	3843
macro avg	0.92	0.93	0.93	3843
weighted avg	0.93	0.93	0.93	3843

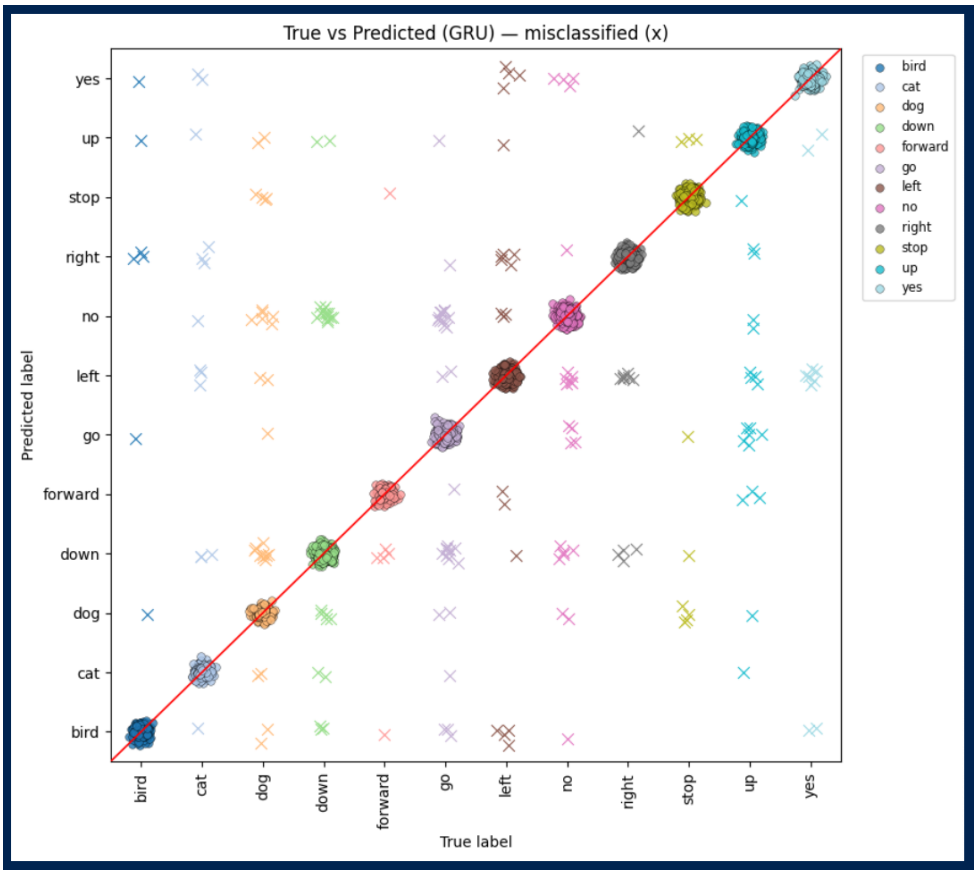
GRU


GRU penultimate features (PCA 2D) — labeled classes; X = misclassified
PC2
bird
cat
dog
down
forward
go
left
no
right
stop
up
yes
misclassified

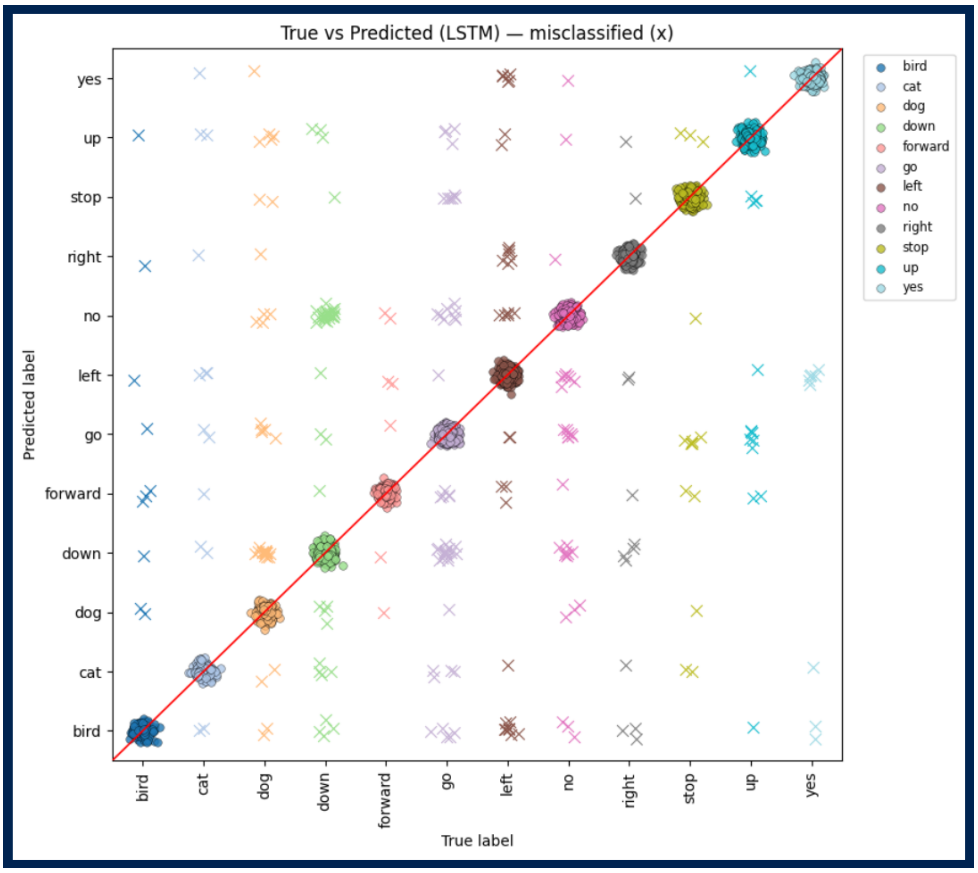
LSTM


LSTM penultimate features (PCA 2D) — labeled classes; X = misclassified
PC1
PC2
bird
cat
dog
down
forward
go
left
no
right
stop
up
yes
misclassified

GRU


True vs Predicted (GRU) — misclassified (x)
Predicted label
True label
bird
cat
dog
down
forward
go
left
no
right
stop
up
yes

LSTM


True vs Predicted (LSTM) — misclassified (x)
Predicted label
True label
bird
cat
dog
down
forward
go
left
no
right
stop
up
yes

Results & Discussion

Learning Curve:

Fast learning occurs early, especially in the GRU model: big drop in both train and val loss in first 3 - 5 epochs - model appears to quickly learn core patterns. Converged by ~ 6-8 epochs: losses flatten after that with little improvement afterwards in the GRU with continued small gains in the LSTM.

Actions for possible gains per Learning Curve results: lower LR and fine-tune, add data/augmentation, or increase model capacity; possibly adjust "patience" variable so use of early stopping occurs during model epoch runs.

Penultimate-Layer: Activations form distinct, class-specific clusters in PCA space, with most misclassified samples lying near class boundaries. The GRU embeddings appear tighter and more separable than the LSTM's, which shows more overlap and mixed regions.

True vs Predicted Scatter Plots: Both plots show most points on the diagonal, so the majority of predictions are mostly correct. The GRU points look slightly tighter with marginally fewer off-diagonal misclassifications, but the two models are very similar overall. For accurate comparative differences refer to precision, recall, and f1 scores.

Overall accuracy:

The model performs very well overall with a test accuracy of ~ 92.8% (reported accuracy 0.93) across 3,843 samples. Most classes have high precision/recall/f1 (typically 0.88 - 0.98); the strongest classes include "right", "yes", and "stop" (f1 ≈ 0.96 - 0.98). Relative weaknesses are small: "dog" shows the lowest recall (0.83, f1 0.88) and "bird" has the lowest precision (0.87, f1 0.91). Macro and weighted averages are ≈ 0.92 - 0.93, indicating a balanced overall performance.

Successes:

The models work. Both GRU and LSTM reached high test accuracy (~ 92 - 96%) depending on the run with clean learning curves and the best-epoch checkpointing in place. PCA of penultimate features shows well-separated class clusters and the true-vs-predicted scatter & confusion matrix show mostly correct classifications with limited, interpretable confusions. Real-time inference pipeline runs and recognizes trained words reliably with minimal unrecognized words.

Challenges:

Reproducibility is a problem and label-order issues were an issue which required fixes (store label order in checkpoints) for the live inference to display correct responses. I believe M1 Mac MPS nondeterminism is the culprit that can still change results/timing across runs. Remaining problems to note: some class imbalances and noisy clips cause specific confusions (e.g., dog/go), VAD and confidence thresholds need tuning for live use, and deployment/performance may vary by hardware (M1 vs. Windows vs Linux desktop GPUs).

Conclusion

Summary: Both RNNs (GRU & LSTM) work with high test accuracy (~92-96%), clear class separation in penultimate features, stable learning curves, and a working real-time inference pipeline.

Major contributions: Deterministic speaker-unique data pipeline + clips.tsv for reproducible splits. Direct comparison of GRU vs LSTM on the 12-word task with checkpointing and early stopping. Lightweight live-inference script (VAD + rolling buffer).

Limitations & future work: Training is lengthly requiring a minimum of 1 hour for the completion of training both models. Reproducibility can still vary on Apple MPS; use full seeding, sorted lists, or CUDA/CPU deterministic runs for exact repeatability. Improve robustness/latency for live use: better VAD, smoothing, and confidence calibration. Try stronger front-ends (1D-conv or small CNN), transformers, or larger GPUs for faster training and possibly higher accuracy. Automate hyperparameter search, per-speaker augmentation, and a small deployment wrapper for real-time accuracy monitoring. Possible faster training option via precomputed mel-spectrogram features.

References

Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition
by: Pete Warden link: **https://arxiv.org/abs/1804.03209**

Guide to Numpy 2nd Edition by: travis E. Oliphant PhD

Introducing Python by: Lubaovic

Automate The Boring Stuff with Python by: Al Sweigart

The Hundred-Page Machine Learning Book by: Burkov

An Introduction to Statistical Learning with Application in Python by: Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani, Jonathan Taylor

Fig 1 & 2): www.researchgate.net

Fig 1 Link: https://www.researchgate.net/publication/357365888/figure/fig10/AS:1105864027049989@1640669938292/Figure-B28-Detailed-GRU-architecture-with-as-trainable-model-parameters-the-weights.png

Fig 2 Link: https://www.researchgate.net/profile/Ahmed-Nait-Chabane/publication/352128261/figure/fig5/AS:1143128109096695O@1666259233292/The-Long-Short-Term-Memory-LSTM-architecture.ppm