🧠 Phase 4: Subvocalization Classification

Single-Channel sEMG → Silent Speech Recognition

Transfer Learning: Mouthing (L3) → Subvocal (L4)

Author: Carl Kho | **Date:** December 2025 | **GPU:** A100 Recommended

1️⃣ Setup & Data Upload

```
# Check GPU
!nvidia-smi
```

```
/bin/bash: line 1: nvidia-smi: command not found
```

```
# Install dependencies
!pip install -q pandas numpy matplotlib seaborn scikit-learn tensorflow scipy
```

```
# Upload your data.zip (contains CSV files)
from google.colab import files
import zipfile
import os

print("📁 Upload your speech-capture.zip file:")
uploaded = files.upload()

# Extract
zip_name = list(uploaded.keys())[0]
with zipfile.ZipFile(zip_name, 'r') as zip_ref:
    zip_ref.extractall('data')

print("\n✅ Extracted files:")
!ls -la data/
```

```
📁 Upload your speech-capture.zip file:
Choose Files speech-capture.zip
speech-capture.zip(application/zip) - 4656542 bytes, last modified: 12/20/2025 - 100% done
Saving speech-capture.zip to speech-capture (2).zip

✅ Extracted files:
total 23840
drwxr-xr-x 2 root root    4096 Dec 19 18:04 .
drwxr-xr-x 1 root root    4096 Dec 19 18:11 ..
-rw-r--r-- 1 root root 1968054 Dec 19 18:11 imagined_data.csv
-rw-r--r-- 1 root root 8854504 Dec 19 18:11 mouthing_data.csv
-rw-r--r-- 1 root root 1988239 Dec 19 18:11 overt_data.csv
-rw-r--r-- 1 root root 9607467 Dec 19 18:11 subvocal_data.csv
-rw-r--r-- 1 root root 1974788 Dec 19 18:11 whisper_data.csv
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import tensorflow as tf
from tensorflow.keras import layers, Model
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
import warnings
warnings.filterwarnings('ignore')

# Set seeds
```

```
np.random.seed(1738)
tf.random.set_seed(1738)

print(f"TensorFlow: {tf.__version__}")
print(f"GPU Available: {tf.config.list_physical_devices('GPU')}")
```

```
TensorFlow: 2.19.0
GPU Available: []
```

2️⃣ Load Data

```
# Find the data directory (handles nested extraction)
import glob

csv_files = glob.glob('data/**/*.csv', recursive=True)
if not csv_files:
    csv_files = glob.glob('data/*.csv')

DATA_DIR = os.path.dirname(csv_files[0]) if csv_files else 'data'
print(f"Data directory: {DATA_DIR}")
print(f"Found CSVs: {csv_files}")
```

```
Data directory: data
Found CSVs: ['data/whisper_data.csv', 'data/mouthing_data.csv', 'data/imagined_data.csv', 'data/subvocal_dat
```

```
def load_spectrum_data(data_dir):
    """Load all motor intensity spectrum CSV files."""
    files = {
        'overt': 'overt_data.csv',
        'whisper': 'whisper_data.csv',
        'mouthing': 'mouthing_data.csv',
        'subvocal': 'subvocal_data.csv',
        'imagined': 'imagined_data.csv'
    }

    data = {}
    for level, filename in files.items():
        filepath = os.path.join(data_dir, filename)
        if os.path.exists(filepath):
            df = pd.read_csv(filepath)
            data[level] = df
            print(f"✅ Loaded {level}: {len(df):,} samples")
        else:
            print(f"⚠️  Missing: {filename}")

    return data

data = load_spectrum_data(DATA_DIR)
```

```
✅ Loaded overt: 108,894 samples
✅ Loaded whisper: 108,157 samples
✅ Loaded mouthing: 515,501 samples
✅ Loaded subvocal: 537,901 samples
✅ Loaded imagined: 107,791 samples
```

```
# Quick look at the data
print("\n📊 Sample data (mouthing):")
data['mouthing'].head()
```

Sample data (mouthing):

	Label	Timestamp	RawValue
0	GHOST	53539	1919
1	GHOST	53542	1919
2	GHOST	53543	1919
3	GHOST	53544	1918
4	GHOST	53545	1918

Due Diligence: Data Quality Assessment

Before modeling, we perform comprehensive diagnostics to verify data quality, class balance, and signal characteristics.

```
# ========================================
# DUE DILIGENCE: Comprehensive Diagnostics
# ========================================
print("=" * 60)
print("DUE DILIGENCE: Data Quality Assessment")
print("=" * 60)

# 1. Class Balance Check
print("\n1 CLASS BALANCE:")
for level_name, df in data.items():
    print(f"\n  {level_name.upper()}:")
    class_counts = df['Label'].value_counts()
    total = len(df)
    for label, count in class_counts.items():
        pct = count / total * 100
        print(f"    {label}: {count:,} samples ({pct:.1f}%)")
```

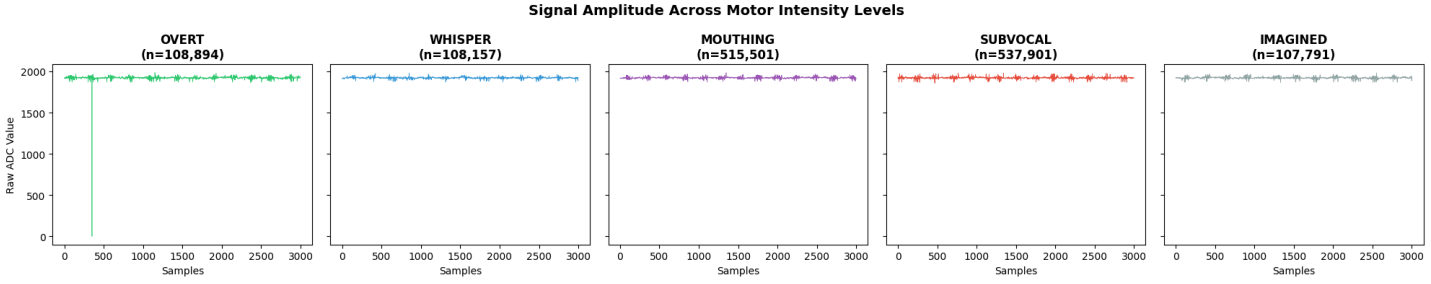
```
============================================================
DUE DILIGENCE: Data Quality Assessment
============================================================

1 CLASS BALANCE:

  OVERT:
    GHOST: 28,126 samples (25.8%)
    LEFT: 26,957 samples (24.8%)
    REST: 26,919 samples (24.7%)
    STOP: 26,892 samples (24.7%)

  WHISPER:
    GHOST: 27,931 samples (25.8%)
    LEFT: 26,767 samples (24.7%)
    STOP: 26,742 samples (24.7%)
    REST: 26,717 samples (24.7%)

  MOUTHING:
    GHOST: 129,354 samples (25.1%)
    STOP: 128,853 samples (25.0%)
    LEFT: 128,703 samples (25.0%)
    REST: 128,591 samples (24.9%)

  SUBVOCAL:
    GHOST: 135,906 samples (25.3%)
    REST: 134,032 samples (24.9%)
    STOP: 133,993 samples (24.9%)
    LEFT: 133,970 samples (24.9%)

  IMAGINED:
    GHOST: 27,791 samples (25.8%)
    LEFT: 26,719 samples (24.8%)
    STOP: 26,647 samples (24.7%)
    REST: 26,634 samples (24.7%)
```

```
# 2. Signal Statistics Comparison
print("\n2️⃣ SIGNAL STATISTICS (Raw ADC values):")
stats_data = []
for level_name, df in data.items():
    stats = {
        'Level': level_name.upper(),
        'Mean': df['RawValue'].mean(),
        'Std': df['RawValue'].std(),
        'Min': df['RawValue'].min(),
        'Max': df['RawValue'].max(),
        'Range': df['RawValue'].max() - df['RawValue'].min()
    }
    stats_data.append(stats)

stats_df = pd.DataFrame(stats_data)
print(stats_df.to_string(index=False))
```

```
2️⃣ SIGNAL STATISTICS (Raw ADC values):
   Level        Mean         Std  Min    Max  Range
   OVERT 1921.366375  12.314980    1   1989   1988
 WHISPER 1921.019065   8.981001 1857   1987    130
MOUTHING 1921.178238   9.752317 1853   1991    138
SUBVOCAL 1921.151398 260.618888   22 192921 192899
IMAGINED 1921.283883   9.552761 1858   1987    129
```

```
# 3. Amplitude Comparison Across Levels (Visual)
print("\n3️⃣ AMPLITUDE COMPARISON ACROSS MOTOR INTENSITY LEVELS:")
fig, axes = plt.subplots(1, 5, figsize=(20, 4), sharey=True)
levels = ['overt', 'whisper', 'mouthing', 'subvocal', 'imagined']
colors = ['#2ecc71', '#3498db', '#9b59b6', '#e74c3c', '#95a5a6']

for ax, level, color in zip(axes, levels, colors):
    if level in data:
        # Take a 3-second sample
        sample = data[level]['RawValue'].values[:3000]
        ax.plot(sample, linewidth=0.5, color=color)
        ax.set_title(f'{level.upper()}\n(n={len(data[level]):,})', fontweight='bold')
        ax.set_xlabel('Samples')

axes[0].set_ylabel('Raw ADC Value')
plt.suptitle('Signal Amplitude Across Motor Intensity Levels', fontsize=14, fontweight='bold')
plt.tight_layout()
plt.savefig('viz_amplitude_comparison.png', dpi=150)
plt.show()
```

```
3️⃣ AMPLITUDE COMPARISON ACROSS MOTOR INTENSITY LEVELS:
```




```
# 4. Sample Duration Distribution
print("\n4️⃣ SAMPLE DURATION PER WORD (Block Lengths):")
for level_name in ['mouthing', 'subvocal']:
    if level_name in data:
        df = data[level_name].copy()
        df['label_change'] = df['Label'] != df['Label'].shift(1)
        df['block_id'] = df['label_change'].cumsum()
        block_lengths = df.groupby('block_id').size()

        print(f"\n  {level_name.upper()}:")
        print(f"    Mean block length: {block_lengths.mean():.0f} samples ({block_lengths.mean()/1000:.2f}s
```

```
        print(f"    Std: {block_lengths.std():.0f} samples")
        print(f"    Min: {block_lengths.min()} | Max: {block_lengths.max()}")
        print(f"    Total blocks: {len(block_lengths)}")
```

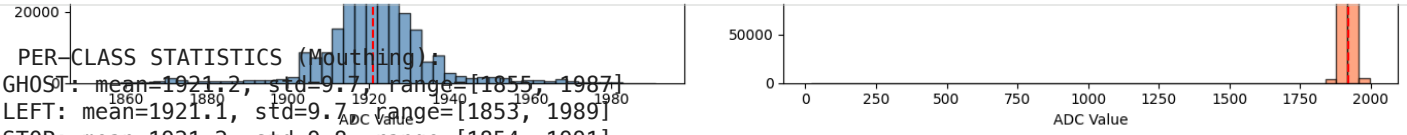
```
4️⃣ SAMPLE DURATION PER WORD (Block Lengths):

  MOUTHING:
    Mean block length: 2578 samples (2.58s)
    Std: 94 samples
    Min: 2499 | Max: 3810
    Total blocks: 200

  SUBVOCAL:
    Mean block length: 2676 samples (2.68s)
    Std: 167 samples
    Min: 935 | Max: 3740
    Total blocks: 201
```

✦ Gemini

```
# 5. ADC Value Distribution
print("\n5️⃣ ADC VALUE DISTRIBUTION (Histogram):")

# FILTER OUTLIERS (Fixes the "weird" x-axis)
# The max value was ~192,000, while valid signals are ~1900.
# We filter out anything > 4000 (safe upper bound).
if 'subvocal' in data:
    initial_len = len(data['subvocal'])
    data['subvocal'] = data['subvocal'][data['subvocal']['RawValue'] < 4000]
    removed = initial_len - len(data['subvocal'])
    if removed > 0:
        print(f"🧹 CLEANUP: Removed {removed} outliers from Subvocal data (Values > 4000).")

fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Mouthing
axes[0].hist(data['mouthing']['RawValue'], bins=50, color='steelblue', alpha=0.7, edgecolor='black')
axes[0].set_title('Mouthing (L3) ADC Distribution', fontweight='bold')
axes[0].set_xlabel('ADC Value')
axes[0].set_ylabel('Frequency')
axes[0].axvline(data['mouthing']['RawValue'].mean(), color='red', linestyle='--', label=f"Mean: {data['mouth
axes[0].legend()

# Subvocal
axes[1].hist(data['subvocal']['RawValue'], bins=50, color='coral', alpha=0.7, edgecolor='black')
axes[1].set_title('Subvocal (L4) ADC Distribution', fontweight='bold')
axes[1].set_xlabel('ADC Value')
axes[1].set_ylabel('Frequency')
axes[1].axvline(data['subvocal']['RawValue'].mean(), color='red', linestyle='--', label=f"Mean: {data['subvo
axes[1].legend()

plt.tight_layout()
plt.savefig('viz_adc_distribution.png', dpi=150)
plt.show()
```

```
5️⃣ ADC VALUE DISTRIBUTION (Histogram):
🧹 CLEANUP: Removed 1 outliers from Subvocal data (Values > 4000).
```


```
# 6. Per-Class Signal Comparison (Mouthing)
print("\n6️⃣ PER-CLASS STATISTICS (Mouthing):")
for label in data['mouthing']['Label'].unique():
    subset = data['mouthing'][data['mouthing']['Label'] == label]['RawValue']
    print(f"  {label}: mean={subset.mean():.1f}, std={subset.std():.1f}, range=[{subset.min()}, {subset.max

print("\n" + "=" * 60)
print("✅ DUE DILIGENCE COMPLETE")
print("=" * 60)
```




```
6️⃣ PER-CLASS STATISTICS (Mouthing):
  GHOST: mean=1921.2, std=9.7, range=[1855, 1987]
  LEFT: mean=1921.1, std=9.7, range=[1853, 1989]
  STOP: mean=1921.2, std=9.8, range=[1854, 1991]
  REST: mean=1921.2, std=9.8, range=[1856, 1989]

============================================================
✅ DUE DILIGENCE COMPLETE
============================================================
```

3️⃣ Preprocessing & Windowing

```
from scipy.signal import butter, filtfilt, iirnotch

def bandpass_filter(signal, fs=1000, lowcut=1.0, highcut=45.0, order=4):
    """Apply Butterworth bandpass filter."""
    nyq = 0.5 * fs
    low = lowcut / nyq
    high = highcut / nyq
    b, a = butter(order, [low, high], btype='band')
    return filtfilt(b, a, signal)

def notch_filter(signal, fs=1000, freq=60.0, Q=30.0):
    """Remove 60Hz power line noise."""
    b, a = iirnotch(freq, Q, fs)
    return filtfilt(b, a, signal)

def preprocess_signal(signal):
    """Full preprocessing pipeline."""
    signal = bandpass_filter(signal)
    signal = notch_filter(signal)
    # Z-score normalize
    return (signal - signal.mean()) / (signal.std() + 1e-8)
```

```
def create_windows(df, window_size=1000, center_offset=1000):
    """
    Create fixed-size windows grouped by label transitions.

    IMPORTANT: Words were vocalized at countdown "2" (middle of 3-sec window).
    So we extract samples [1000:2000] to capture the actual articulation.

    Args:
        window_size: Size of window to extract (1000 = 1 second @ 1000Hz)
        center_offset: Where the word starts in the raw block (1000 = at 1 second)
    """
    windows_X = []
    windows_y = []

    # Find label transitions
    df['label_change'] = df['Label'] != df['Label'].shift(1)
    df['block_id'] = df['label_change'].cumsum()
```

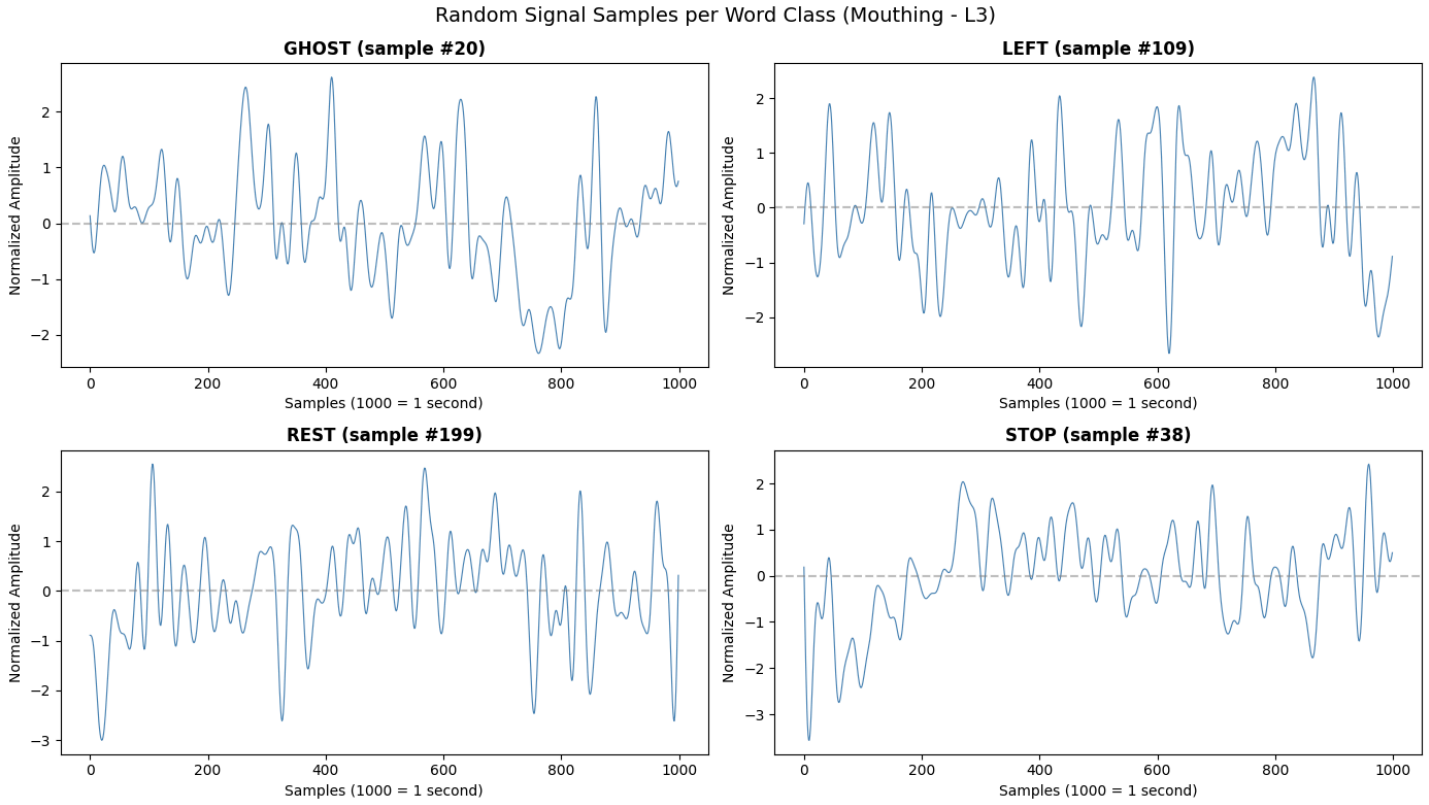
```
    for block_id, block in df.groupby('block_id'):
        label = block['Label'].iloc[0]
        signal = block['RawValue'].values

        # Extract MIDDLE portion where word was actually spoken
        # Words vocalized at countdown "2" = samples 1000-2000
        if len(signal) >= center_offset + window_size:
            signal = signal[center_offset:center_offset + window_size]
        elif len(signal) >= window_size:
            # Fallback: take last window_size samples
            signal = signal[-window_size:]
        else:
            # Pad if too short
            pad_size = window_size - len(signal)
            signal = np.pad(signal, (0, pad_size), mode='mean')

        # Preprocess
        try:
            signal = preprocess_signal(signal)
            windows_X.append(signal.reshape(-1, 1))
            windows_y.append(label)
        except:
            continue  # Skip problematic windows

    return np.array(windows_X), np.array(windows_y)

print("Creating windows (extracting MIDDLE 1-second where word was spoken)...")
X_mouthing, y_mouthing = create_windows(data['mouthing'])
X_subvocal, y_subvocal = create_windows(data['subvocal'])

print(f"\n📊 Mouthing (L3 - Training): {X_mouthing.shape}")
print(f"📊 Subvocal (L4 - Testing): {X_subvocal.shape}")
```

```
Creating windows (extracting MIDDLE 1-second where word was spoken)...

📊 Mouthing (L3 - Training): (200, 1000, 1)
📊 Subvocal (L4 - Testing): (201, 1000, 1)
```

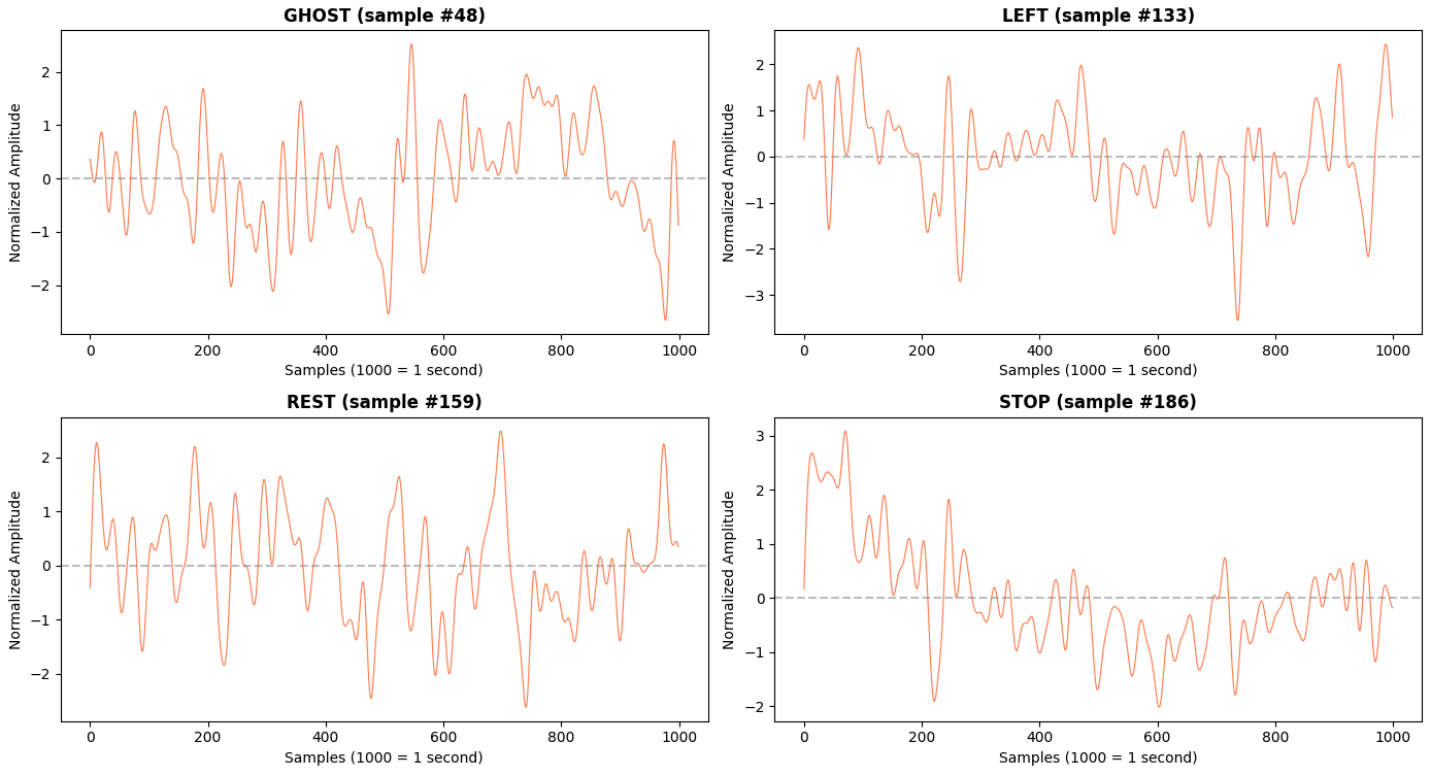
```
# 📊 Visualize random samples for each class
print("\n📊 Random samples per class (Mouthing - L3):")
unique_classes = np.unique(y_mouthing)
fig, axes = plt.subplots(2, 2, figsize=(14, 8))
axes = axes.flatten()

for i, cls in enumerate(unique_classes):
    # Find indices for this class
    cls_indices = np.where(y_mouthing == cls)[0]
    if len(cls_indices) > 0:
        # Pick random sample
        rand_idx = np.random.choice(cls_indices)
        signal = X_mouthing[rand_idx].flatten()

        axes[i].plot(signal, linewidth=0.8, color='steelblue')
        axes[i].set_title(f'{cls} (sample #{rand_idx})', fontsize=12, fontweight='bold')
        axes[i].set_xlabel('Samples (1000 = 1 second)')
        axes[i].set_ylabel('Normalized Amplitude')
        axes[i].axhline(y=0, color='gray', linestyle='--', alpha=0.5)

plt.suptitle('Random Signal Samples per Word Class (Mouthing - L3)', fontsize=14)
plt.tight_layout()
plt.savefig('viz_random_samples_mouthing.png', dpi=150)
plt.show()

# Same for subvocal
print("\n📊 Random samples per class (Subvocal - L4):")
fig, axes = plt.subplots(2, 2, figsize=(14, 8))
axes = axes.flatten()

for i, cls in enumerate(unique_classes):
```

```
    cls_indices = np.where(y_subvocal == cls)[0]
    if len(cls_indices) > 0:
        rand_idx = np.random.choice(cls_indices)
        signal = X_subvocal[rand_idx].flatten()

        axes[i].plot(signal, linewidth=0.8, color='coral')
        axes[i].set_title(f'{cls} (sample #{rand_idx})', fontsize=12, fontweight='bold')
        axes[i].set_xlabel('Samples (1000 = 1 second)')
        axes[i].set_ylabel('Normalized Amplitude')
        axes[i].axhline(y=0, color='gray', linestyle='--', alpha=0.5)

plt.suptitle('Random Signal Samples per Word Class (Subvocal - L4)', fontsize=14)
plt.tight_layout()
plt.savefig('viz_random_samples_subvocal.png', dpi=150)
plt.show()
```

Random samples per class (Mouthing – L3):



Random samples per class (Subvocal – L4):




```
# Encode labels
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
le.fit(np.concatenate([y_mouthing, y_subvocal]))

y_mouthing_enc = le.transform(y_mouthing)
```

```
y_subvocal_enc = le.transform(y_subvocal)

print(f"Classes: {le.classes_}")
print(f"Label mapping: {dict(zip(le.classes_, range(len(le.classes_))))}")
```

```
Classes: ['GHOST' 'LEFT' 'REST' 'STOP']
Label mapping: {np.str_('GHOST'): 0, np.str_('LEFT'): 1, np.str_('REST'): 2, np.str_('STOP'): 3}
```

```
# Train/Val split on mouthing (source domain)
X_train, X_val, y_train, y_val = train_test_split(
    X_mouthing, y_mouthing_enc,
    test_size=0.15,
    random_state=42,
    stratify=y_mouthing_enc
)

# Test set is subvocal (target domain)
X_test, y_test = X_subvocal, y_subvocal_enc

print(f"\n📊 Train: {X_train.shape}")
print(f"📊 Val: {X_val.shape}")
print(f"📊 Test (L4): {X_test.shape}")
```

```
📊 Train: (170, 1000, 1)
📊 Val: (30, 1000, 1)
📊 Test (L4): (201, 1000, 1)
```

4️⃣ Feature Extraction (EXTENDED for better accuracy)

```
# ACCURACY IMPROVEMENT #1: Extended Feature Set
def extract_features_extended(X):
    """
    Extract EXTENDED features for better discrimination.

    Features:
    - Time domain: MAV, ZCR, SD, MAX, RMS, Waveform Length
    - Temporal: Energy in 4 quarters
    - Frequency: Dominant frequency, spectral centroid

    Total: 14 features per window
    """
    from scipy.fft import fft

    features = []
    for window in X:
        signal = window.flatten()
        n = len(signal)

        # Time domain features
        mav = np.mean(np.abs(signal))
        zcr = np.sum(np.diff(np.sign(signal)) != 0)
        sd = np.std(signal)
        max_amp = np.max(np.abs(signal))
        rms = np.sqrt(np.mean(signal**2))
        waveform_length = np.sum(np.abs(np.diff(signal)))

        # Temporal features (quarters)
        e1 = np.mean(np.abs(signal[:n//4]))
        e2 = np.mean(np.abs(signal[n//4:n//2]))
        e3 = np.mean(np.abs(signal[n//2:3*n//4]))
        e4 = np.mean(np.abs(signal[3*n//4:]))

        # Frequency features (FFT)
        fft_vals = np.abs(fft(signal))[:n//2]  # Only positive frequencies
        freqs = np.linspace(0, 500, n//2)  # 0-500Hz for 1000Hz sampling

        # Dominant frequency
```

```
        dom_freq_idx = np.argmax(fft_vals[1:]) + 1  # Skip DC
        dom_freq = freqs[dom_freq_idx]

        # Spectral centroid
        spectral_centroid = np.sum(freqs * fft_vals) / (np.sum(fft_vals) + 1e-8)

        # Spectral energy in speech band (1-45Hz)
        speech_band_mask = (freqs >= 1) & (freqs <= 45)
        speech_band_energy = np.sum(fft_vals[speech_band_mask])

        features.append([
            mav, zcr, sd, max_amp, rms, waveform_length,
            e1, e2, e3, e4,
            dom_freq, spectral_centroid, speech_band_energy,
            e2 - e1  # Onset indicator
        ])

    return np.array(features)

print("Extracting EXTENDED features (14 features)...")
X_train_feat = extract_features_extended(X_train)
X_val_feat = extract_features_extended(X_val)
X_test_feat = extract_features_extended(X_test)

print(f"Feature shapes: Train {X_train_feat.shape}, Val {X_val_feat.shape}, Test {X_test_feat.shape}")
```

```
Extracting EXTENDED features (14 features)...
Feature shapes: Train (170, 14), Val (30, 14), Test (201, 14)
```

```
# ACCURACY IMPROVEMENT #2: Data Augmentation
def augment_data(X, y, factor=3):
    """
    Augment training data with jitter, scaling, and time shift.
    """
    X_aug = [X]
    y_aug = [y]

    for _ in range(factor - 1):
        X_new = []
        for window in X:
            aug = window.copy()

            # Random jitter
            aug += np.random.normal(0, 0.05, aug.shape)

            # Random scaling
            aug *= np.random.uniform(0.9, 1.1)

            # Random time shift (circular)
            shift = np.random.randint(-50, 50)
            aug = np.roll(aug, shift, axis=0)

            X_new.append(aug)

        X_aug.append(np.array(X_new))
        y_aug.append(y)

    return np.vstack(X_aug), np.hstack(y_aug)

print("\nAugmenting training data (3x)...")
X_train_aug, y_train_aug = augment_data(X_train, y_train, factor=3)
X_train_feat_aug = extract_features_extended(X_train_aug)
print(f"Augmented: {X_train_feat.shape} → {X_train_feat_aug.shape}")
```

```

Augmenting training data (3x)...
Augmented: (170, 14) → (510, 14)
```

5️⃣ Random Forest Baseline

```
print("🌲 Training Random Forest (with AUGMENTED data)...")

rf = RandomForestClassifier(
    n_estimators=200,  # More trees for better accuracy
    max_depth=20,      # Prevent overfitting
    min_samples_split=5,
    random_state=1738,
    n_jobs=-1
)

# Train on AUGMENTED data
rf.fit(X_train_feat_aug, y_train_aug)

# Evaluate on source domain (val)
y_val_pred = rf.predict(X_val_feat)
val_acc = accuracy_score(y_val, y_val_pred)

# Evaluate on target domain (test = L4)
y_test_pred = rf.predict(X_test_feat)
test_acc = accuracy_score(y_test, y_test_pred)

print(f"\n✅ Val Accuracy (L3): {val_acc:.4f}")
print(f"✅ Test Accuracy (L4): {test_acc:.4f}")
print(f"📉 Transfer Gap: {val_acc - test_acc:.4f}")
```

```
🌲 Training Random Forest (with AUGMENTED data)...

✅ Val Accuracy (L3): 0.4667
✅ Test Accuracy (L4): 0.2239
📉 Transfer Gap: 0.2428
```

```
# Compare: Would non-augmented do better?
print("\n📊 Ablation: Augmented vs Non-Augmented:")
rf_no_aug = RandomForestClassifier(n_estimators=200, max_depth=20, random_state=1738, n_jobs=-1)
rf_no_aug.fit(X_train_feat, y_train)
no_aug_acc = accuracy_score(y_test, rf_no_aug.predict(X_test_feat))
print(f"  Without augmentation: {no_aug_acc:.4f}")
print(f"  With augmentation:    {test_acc:.4f}")
print(f"  Improvement: {(test_acc - no_aug_acc)*100:+.2f}%")
```

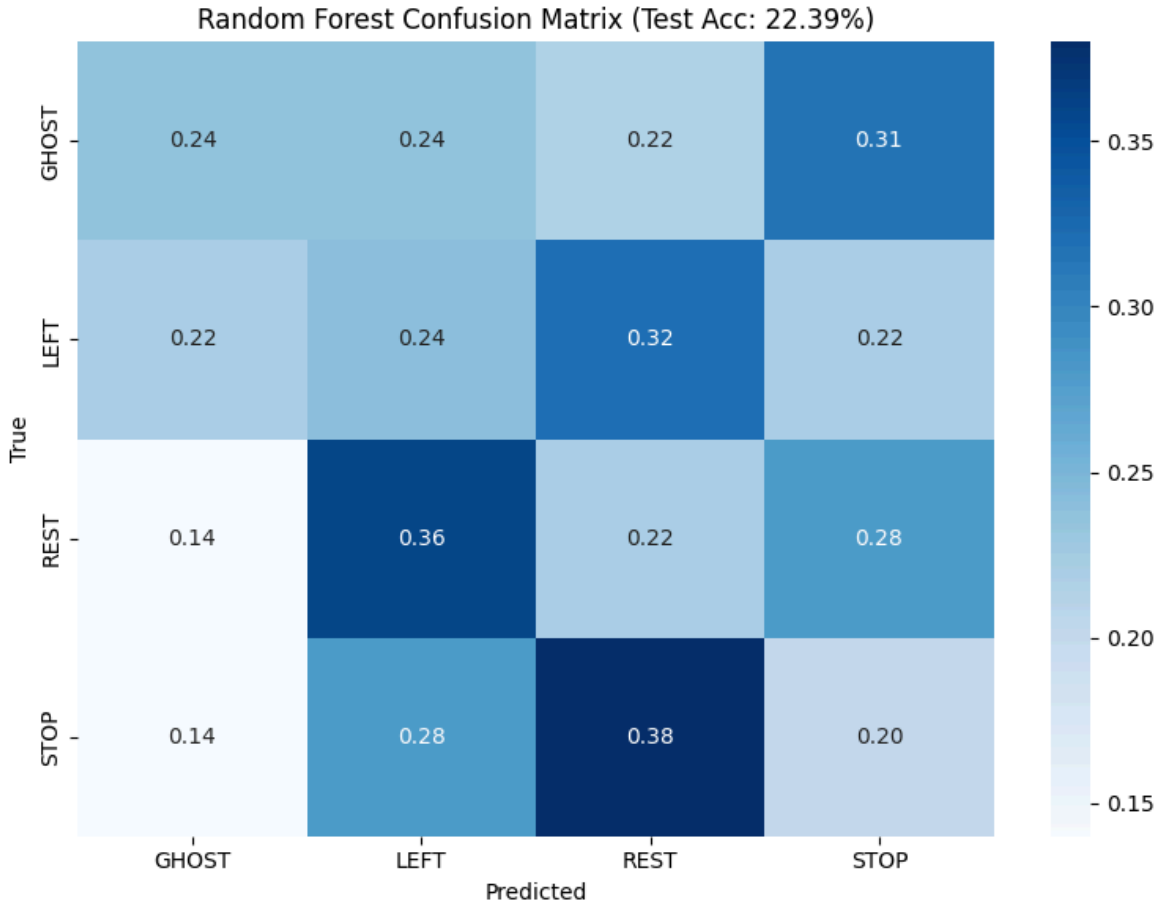
```
📊 Ablation: Augmented vs Non-Augmented:
  Without augmentation: 0.2338
  With augmentation:    0.2239
  Improvement: -1.00%
```

```
# Classification Report
print("\n📊 Classification Report (Test - L4):")
print(classification_report(y_test, y_test_pred, target_names=le.classes_))
```

```
📊 Classification Report (Test - L4):
              precision    recall  f1-score   support

       GHOST       0.32      0.24      0.27        51
        LEFT       0.21      0.24      0.23        50
        REST       0.19      0.22      0.21        50
        STOP       0.20      0.20      0.20        50

    accuracy                           0.22       201
   macro avg       0.23      0.22      0.23       201
weighted avg       0.23      0.22      0.23       201
```

```
# Confusion Matrix
plt.figure(figsize=(8, 6))
```

```
cm = confusion_matrix(y_test, y_test_pred, normalize='true')
sns.heatmap(cm, annot=True, fmt='.2f', cmap='Blues',
            xticklabels=le.classes_, yticklabels=le.classes_)
plt.title(f'Random Forest Confusion Matrix (Test Acc: {test_acc:.2%})')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.tight_layout()
plt.savefig('rf_confusion_matrix.png', dpi=150)
plt.show()
```



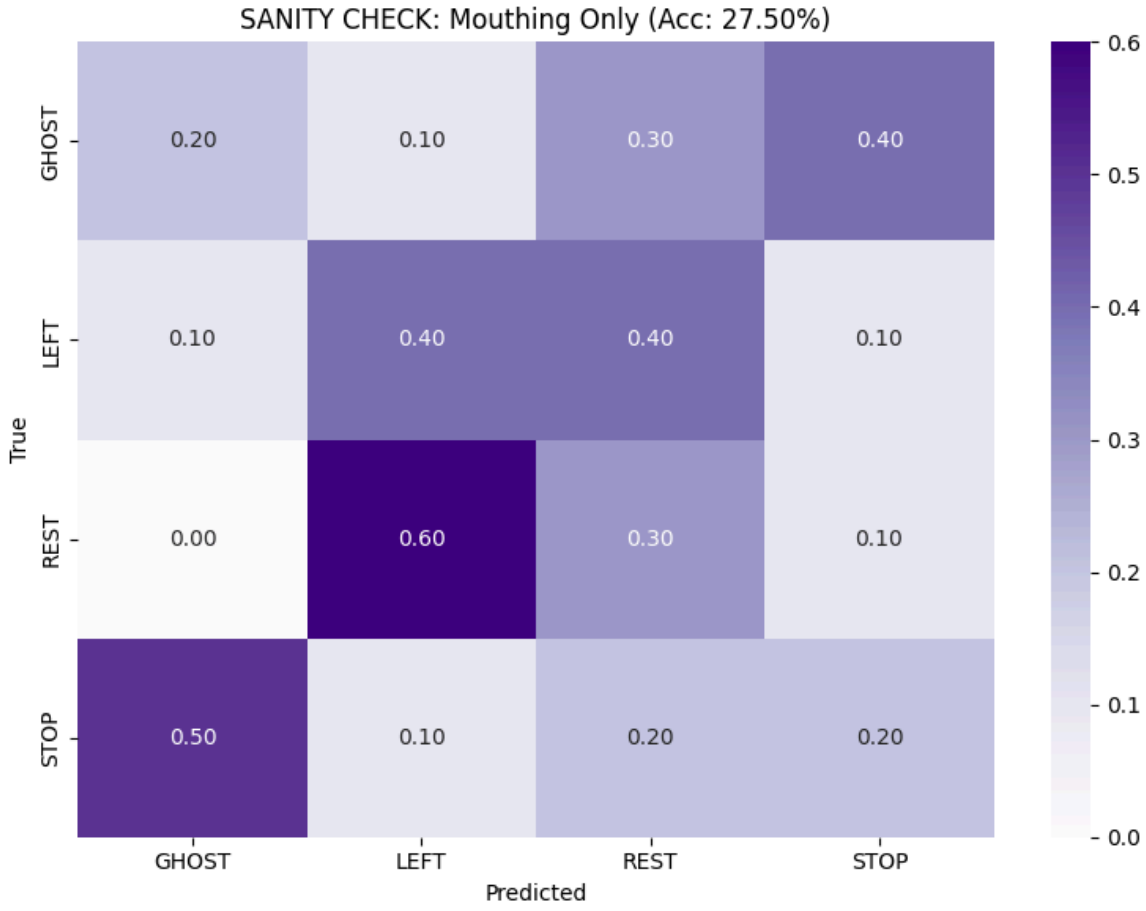
🔬 SANITY CHECK: Same-Domain Classification

Before attempting cross-domain transfer (L3→L4), let's verify that we can classify words **within the same domain**. This tells us: "Is the signal even classifiable with our hardware?"

```
# ========================================
# 🔬 SANITY CHECK: Same-Domain Classification
# ========================================
print("=" * 60)
print("🔬 SANITY CHECK: Train & Test on MOUTHING ONLY (same domain)")
print("=" * 60)

# Split mouthing data properly (train/test from same source)
X_m_train, X_m_test, y_m_train, y_m_test = train_test_split(
    X_mouthing, y_mouthing_enc,
    test_size=0.20,
    random_state=42,
    stratify=y_mouthing_enc
)

# Extract features
# Use the extended feature extractor defined previously
X_m_train_feat = extract_features_extended(X_m_train)
X_m_test_feat = extract_features_extended(X_m_test)
```

```
# Train RF on mouthing only
rf_sanity = RandomForestClassifier(n_estimators=100, random_state=1738, n_jobs=-1)
rf_sanity.fit(X_m_train_feat, y_m_train)

# Evaluate same-domain
y_m_pred = rf_sanity.predict(X_m_test_feat)
sanity_acc = accuracy_score(y_m_test, y_m_pred)

print(f"\n✅ Same-Domain Accuracy (L3→L3): {sanity_acc:.4f}")
print(f"   (This is the 'ceiling' - best we can do)")
print(f"\n📊 Classification Report (L3 only):")
print(classification_report(y_m_test, y_m_pred, target_names=le.classes_))

# Confusion matrix
plt.figure(figsize=(8, 6))
cm = confusion_matrix(y_m_test, y_m_pred, normalize='true')
sns.heatmap(cm, annot=True, fmt='.2f', cmap='Purples',
            xticklabels=le.classes_, yticklabels=le.classes_)
plt.title(f'SANITY CHECK: Mouthing Only (Acc: {sanity_acc:.2%})')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.tight_layout()
plt.savefig('sanity_check_mouthing.png', dpi=150)
plt.show()

print("\n" + "=" * 60)
print("INTERPRETATION:")
print("  > 70%: Signal is good! Transfer L3→L4 is the hard part")
print("  < 50%: Hardware/signal issue - words may not be distinguishable")
print("=" * 60)
```

```
============================================================
🔬 SANITY CHECK: Train & Test on MOUTHING ONLY (same domain)
============================================================

✅ Same-Domain Accuracy (L3→L3): 0.2750
   (This is the 'ceiling' - best we can do)

📊 Classification Report (L3 only):
              precision    recall  f1-score   support

       GHOST       0.25      0.20      0.22        10
        LEFT       0.33      0.40      0.36        10
        REST       0.25      0.30      0.27        10
        STOP       0.25      0.20      0.22        10

    accuracy                           0.28        40
   macro avg       0.27      0.28      0.27        40
weighted avg       0.27      0.28      0.27        40
```




```
============================================================
INTERPRETATION:
  > 70%: Signal is good! Transfer L3→L4 is the hard part
  < 50%: Hardware/signal issue - words may not be distinguishable
============================================================
```

6 MaxCRNN (Deep Learning)

```
def inception_block(x, filters):
    """1D Inception block with parallel convolutions."""
    conv1 = layers.Conv1D(filters, 1, padding='same', activation='relu')(x)
    conv3 = layers.Conv1D(filters, 3, padding='same', activation='relu')(x)
    conv5 = layers.Conv1D(filters, 5, padding='same', activation='relu')(x)
    pool = layers.MaxPooling1D(3, strides=1, padding='same')(x)
    pool = layers.Conv1D(filters, 1, padding='same', activation='relu')(pool)
    return layers.Concatenate()([conv1, conv3, conv5, pool])

def build_maxcrnn(input_shape, n_classes):
```

```
    """Build MaxCRNN: Inception + Bi-LSTM + Attention"""
    inputs = layers.Input(shape=input_shape)

    # Inception blocks
    x = inception_block(inputs, filters=32)
    x = layers.BatchNormalization()(x)
    x = layers.Dropout(0.3)(x)
    x = layers.MaxPooling1D(2)(x)

    x = inception_block(x, filters=64)
    x = layers.BatchNormalization()(x)
    x = layers.Dropout(0.3)(x)
    x = layers.MaxPooling1D(2)(x)

    # Bi-LSTM
    x = layers.Bidirectional(layers.LSTM(64, return_sequences=True))(x)

    # Multi-Head Attention
    x = layers.MultiHeadAttention(num_heads=4, key_dim=16)(x, x)

    # Classification head
    x = layers.GlobalAveragePooling1D()(x)
    x = layers.Dense(32, activation='relu')(x)
    x = layers.Dropout(0.5)(x)
    outputs = layers.Dense(n_classes, activation='softmax')(x)

    return Model(inputs, outputs, name='MaxCRNN')

# Build model
n_classes = len(le.classes_)
model = build_maxcrnn(input_shape=(X_train.shape[1], 1), n_classes=n_classes)
model.summary()
```

Model: "MaxCRNN"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 1000, 1)	0	–
max_pooling1d (MaxPooling1D)	(None, 1000, 1)	0	input_layer[0][0]
conv1d (Conv1D)	(None, 1000, 32)	64	input_layer[0][0]
conv1d_1 (Conv1D)	(None, 1000, 32)	128	input_layer[0][0]
conv1d_2 (Conv1D)	(None, 1000, 32)	192	input_layer[0][0]
conv1d_3 (Conv1D)	(None, 1000, 32)	64	max_pooling1d[0]…
concatenate (Concatenate)	(None, 1000, 128)	0	conv1d[0][0], conv1d_1[0][0], conv1d_2[0][0], conv1d_3[0][0]
batch_normalization (BatchNormalizatio…	(None, 1000, 128)	512	concatenate[0][0]
dropout (Dropout)	(None, 1000, 128)	0	batch_normalizat…
max_pooling1d_1 (MaxPooling1D)	(None, 500, 128)	0	dropout[0][0]
max_pooling1d_2 (MaxPooling1D)	(None, 500, 128)	0	max_pooling1d_1[…
conv1d_4 (Conv1D)	(None, 500, 64)	8,256	max_pooling1d_1[…
conv1d_5 (Conv1D)	(None, 500, 64)	24,640	max_pooling1d_1[…
conv1d_6 (Conv1D)	(None, 500, 64)	41,024	max_pooling1d_1[…
conv1d_7 (Conv1D)	(None, 500, 64)	8,256	max_pooling1d_2[…
concatenate_1 (Concatenate)	(None, 500, 256)	0	conv1d_4[0][0], conv1d_5[0][0], conv1d_6[0][0], conv1d_7[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 500, 256)	1,024	concatenate_1[0]…
dropout_1 (Dropout)	(None, 500, 256)	0	batch_normalizat…
max_pooling1d_3 (MaxPooling1D)	(None, 250, 256)	0	dropout_1[0][0]
bidirectional (Bidirectional)	(None, 250, 128)	164,352	max_pooling1d_3[…
multi_head_attenti… (MultiHeadAttentio…	(None, 250, 128)	33,088	bidirectional[0]… bidirectional[0]…
global_average_poo… (GlobalAveragePool…	(None, 128)	0	multi_head_atten…
dense (Dense)	(None, 32)	4,128	global_average_p…
dropout_3 (Dropout)	(None, 32)	0	dense[0][0]
dense_1 (Dense)	(None, 4)	132	dropout_3[0][0]

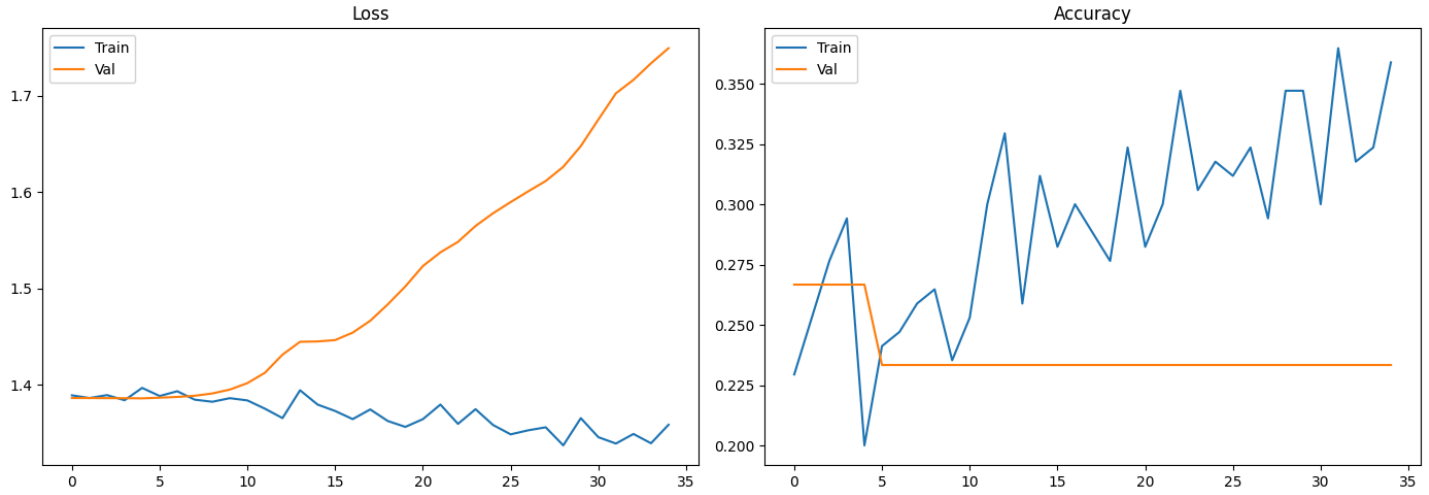
Total params: 285,860 (1.09 MB)
Trainable params: 285,092 (1.09 MB)
Non-trainable params: 768 (3.00 KB)

```
# Compile
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.0005),
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
```

```
)

# Callbacks
callbacks = [
    EarlyStopping(monitor='val_loss', patience=30, restore_best_weights=True),
    ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=10, min_lr=1e-6)
]

print("🚀 Training MaxCRNN...")
history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=200,
    batch_size=32,
    callbacks=callbacks,
    verbose=1
)
```

```
6/6 ━━━━━━━━━━━━━━━━━━━━ 11s 2s/step - accuracy: 0.2588 - loss: 1.3893 - val_accuracy: 0.2333 - val_loss: 1
Epoch 8/200
6/6 ━━━━━━━━━━━━━━━━━━━━ 10s 2s/step - accuracy: 0.2540 - loss: 1.3814 - val_accuracy: 0.2333 - val_loss: 1
Epoch 9/200
6/6 ━━━━━━━━━━━━━━━━━━━━ 8s 1s/step - accuracy: 0.2816 - loss: 1.3814 - val_accuracy: 0.2333 - val_loss: 1.
Epoch 10/200
6/6 ━━━━━━━━━━━━━━━━━━━━ 10s 1s/step - accuracy: 0.2288 - loss: 1.3875 - val_accuracy: 0.2333 - val_loss: 1
Epoch 11/200
6/6 ━━━━━━━━━━━━━━━━━━━━ 10s 2s/step - accuracy: 0.2516 - loss: 1.3849 - val_accuracy: 0.2333 - val_loss: 1
Epoch 12/200
6/6 ━━━━━━━━━━━━━━━━━━━━ 14s 2s/step - accuracy: 0.3025 - loss: 1.3736 - val_accuracy: 0.2333 - val_loss: 1
Epoch 13/200
6/6 ━━━━━━━━━━━━━━━━━━━━ 15s 1s/step - accuracy: 0.3420 - loss: 1.3695 - val_accuracy: 0.2333 - val_loss: 1
Epoch 14/200
6/6 ━━━━━━━━━━━━━━━━━━━━ 13s 2s/step - accuracy: 0.2147 - loss: 1.3919 - val_accuracy: 0.2333 - val_loss: 1
Epoch 15/200
6/6 ━━━━━━━━━━━━━━━━━━━━ 12s 2s/step - accuracy: 0.3194 - loss: 1.3802 - val_accuracy: 0.2333 - val_loss: 1
Epoch 16/200
6/6 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.2822 - loss: 1.3695 - val_accuracy: 0.2333 - val_loss: 1.
Epoch 17/200
6/6 ━━━━━━━━━━━━━━━━━━━━ 7s 1s/step - accuracy: 0.2849 - loss: 1.3687 - val_accuracy: 0.2333 - val_loss: 1.
Epoch 18/200
6/6 ━━━━━━━━━━━━━━━━━━━━ 9s 1s/step - accuracy: 0.2623 - loss: 1.3777 - val_accuracy: 0.2333 - val_loss: 1.
Epoch 19/200
6/6 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.2811 - loss: 1.3589 - val_accuracy: 0.2333 - val_loss: 1.
Epoch 20/200
6/6 ━━━━━━━━━━━━━━━━━━━━ 7s 1s/step - accuracy: 0.3255 - loss: 1.3607 - val_accuracy: 0.2333 - val_loss: 1.
Epoch 21/200
6/6 ━━━━━━━━━━━━━━━━━━━━ 10s 2s/step - accuracy: 0.2865 - loss: 1.3669 - val_accuracy: 0.2333 - val_loss: 1
Epoch 22/200
6/6 ━━━━━━━━━━━━━━━━━━━━ 11s 2s/step - accuracy: 0.2926 - loss: 1.3748 - val_accuracy: 0.2333 - val_loss: 1
Epoch 23/200
6/6 ━━━━━━━━━━━━━━━━━━━━ 20s 1s/step - accuracy: 0.3446 - loss: 1.3579 - val_accuracy: 0.2333 - val_loss: 1
Epoch 24/200
6/6 ━━━━━━━━━━━━━━━━━━━━ 11s 2s/step - accuracy: 0.2995 - loss: 1.3648 - val_accuracy: 0.2333 - val_loss: 1
Epoch 25/200
6/6 ━━━━━━━━━━━━━━━━━━━━ 11s 2s/step - accuracy: 0.3100 - loss: 1.3510 - val_accuracy: 0.2333 - val_loss: 1
Epoch 26/200
6/6 ━━━━━━━━━━━━━━━━━━━━ 19s 1s/step - accuracy: 0.3110 - loss: 1.3485 - val_accuracy: 0.2333 - val_loss: 1
Epoch 27/200
6/6 ━━━━━━━━━━━━━━━━━━━━ 12s 2s/step - accuracy: 0.3055 - loss: 1.3583 - val_accuracy: 0.2333 - val_loss: 1
Epoch 28/200
6/6 ━━━━━━━━━━━━━━━━━━━━ 28s 3s/step - accuracy: 0.2863 - loss: 1.3584 - val_accuracy: 0.2333 - val_loss: 1
```

```
# Training curves
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

axes[0].plot(history.history['loss'], label='Train')
axes[0].plot(history.history['val_loss'], label='Val')
axes[0].set_title('Loss')
axes[0].legend()

axes[1].plot(history.history['accuracy'], label='Train')
axes[1].plot(history.history['val_accuracy'], label='Val')
axes[1].set_title('Accuracy')
axes[1].legend()

plt.tight_layout()
plt.savefig('maxcrnn_training_curves.png', dpi=150)
plt.show()
```



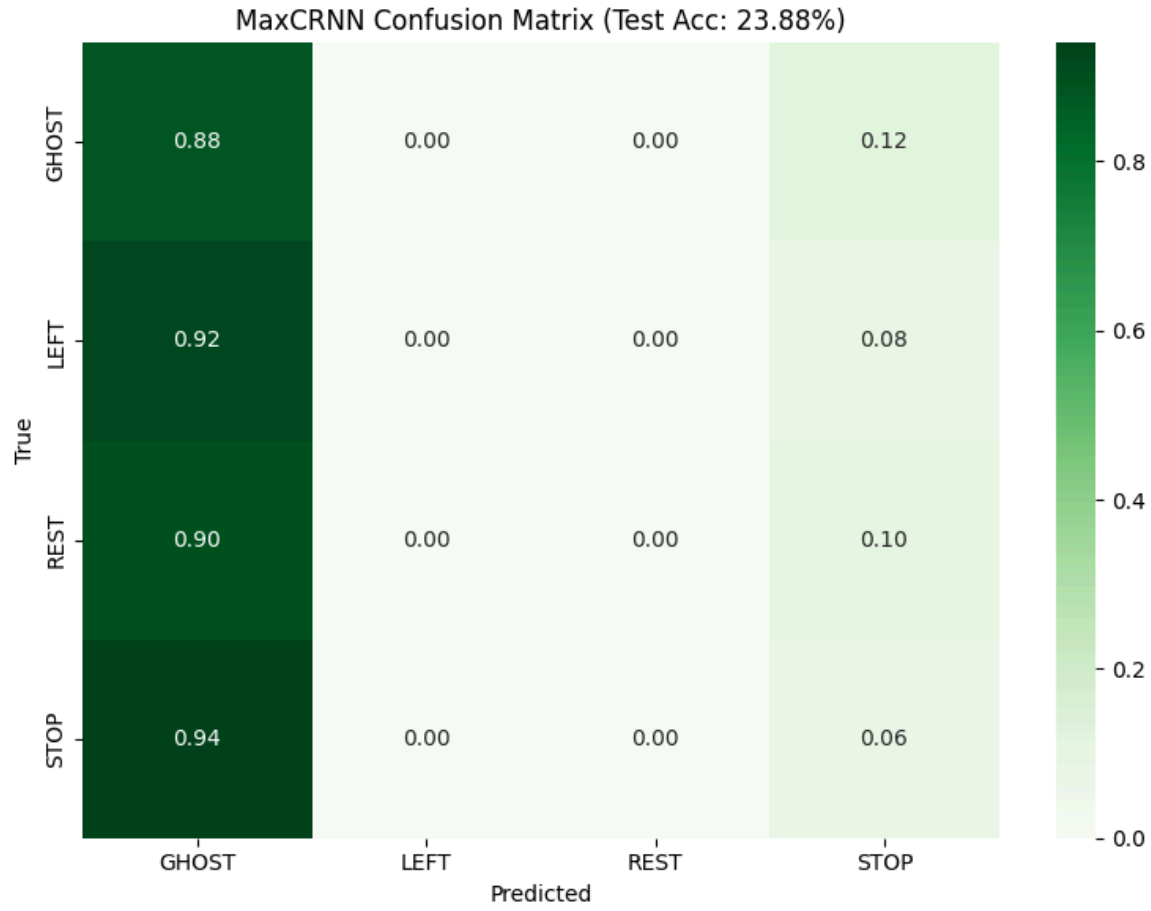

```
# Evaluate MaxCRNN
print("\n📊 MaxCRNN Evaluation:")

# Val (L3)
val_loss, val_acc_nn = model.evaluate(X_val, y_val, verbose=0)
print(f"Val Accuracy (L3): {val_acc_nn:.4f}")

# Test (L4)
test_loss, test_acc_nn = model.evaluate(X_test, y_test, verbose=0)
print(f"Test Accuracy (L4): {test_acc_nn:.4f}")
print(f"Transfer Gap: {val_acc_nn - test_acc_nn:.4f}")
```

```
📊 MaxCRNN Evaluation:
Val Accuracy (L3): 0.2667
Test Accuracy (L4): 0.2388
Transfer Gap: 0.0279
```

```
# MaxCRNN Confusion Matrix
y_test_pred_nn = np.argmax(model.predict(X_test), axis=1)

plt.figure(figsize=(8, 6))
cm = confusion_matrix(y_test, y_test_pred_nn, normalize='true')
sns.heatmap(cm, annot=True, fmt='.2f', cmap='Greens',
            xticklabels=le.classes_, yticklabels=le.classes_)
plt.title(f'MaxCRNN Confusion Matrix (Test Acc: {test_acc_nn:.2%})')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.tight_layout()
plt.savefig('maxcrnn_confusion_matrix.png', dpi=150)
plt.show()
```

```
print("\n📊 Classification Report (MaxCRNN - L4):")
print(classification_report(y_test, y_test_pred_nn, target_names=le.classes_))
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 3s 413ms/step
```



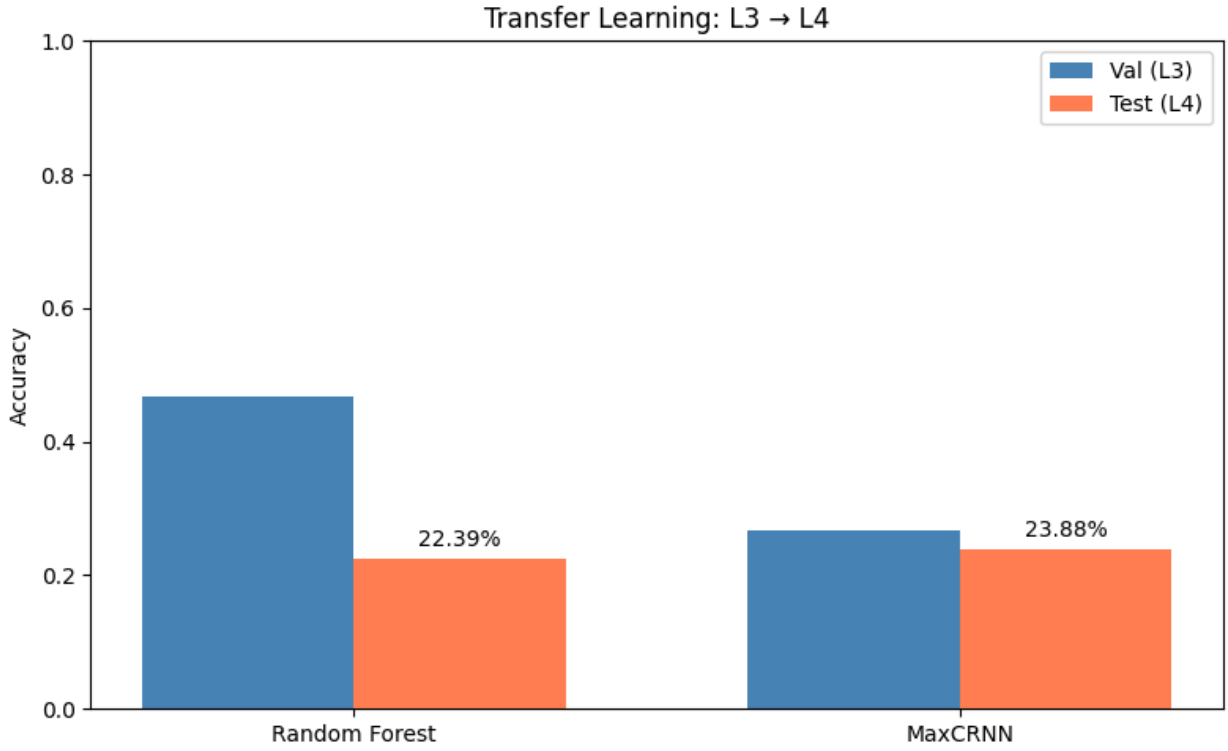

```
📊 Classification Report (MaxCRNN - L4):
              precision    recall  f1-score   support

       GHOST       0.25      0.88      0.38        51
        LEFT       0.00      0.00      0.00        50
        REST       0.00      0.00      0.00        50
        STOP       0.17      0.06      0.09        50

    accuracy                           0.24       201
   macro avg       0.10      0.24      0.12       201
weighted avg       0.10      0.24      0.12       201
```

7️⃣ Model Comparison

```
# Compare models
rf_test_acc = accuracy_score(y_test, y_test_pred)
nn_test_acc = accuracy_score(y_test, y_test_pred_nn)

results = pd.DataFrame({
    'Model': ['Random Forest', 'MaxCRNN'],
    'Test Accuracy (L4)': [rf_test_acc, nn_test_acc],
    'Val Accuracy (L3)': [accuracy_score(y_val, rf.predict(X_val_feat)), val_acc_nn]
})

print("\n🏆 Model Comparison:")
print(results)

# Bar chart
plt.figure(figsize=(8, 5))
x = np.arange(2)
width = 0.35
plt.bar(x - width/2, results['Val Accuracy (L3)'], width, label='Val (L3)', color='steelblue')
```

```
plt.bar(x + width/2, results['Test Accuracy (L4)'], width, label='Test (L4)', color='coral')
plt.xticks(x, results['Model'])
plt.ylabel('Accuracy')
plt.title('Transfer Learning: L3 → L4')
plt.legend()
plt.ylim(0, 1)
for i, v in enumerate(results['Test Accuracy (L4)']):
    plt.text(i + width/2, v + 0.02, f'{v:.2%}', ha='center')
plt.tight_layout()
plt.savefig('model_comparison.png', dpi=150)
plt.show()
```

```
🏆 Model Comparison:
          Model  Test Accuracy (L4)  Val Accuracy (L3)
0  Random Forest            0.223881           0.466667
1        MaxCRNN            0.238806           0.266667
```



8️⃣ Save Models & Download

```
# Save Random Forest
import pickle
with open('random_forest_phase4.pkl', 'wb') as f:
    pickle.dump(rf, f)
print("✅ Saved: random_forest_phase4.pkl")

# Save MaxCRNN
model.save('maxcrnn_phase4.keras')
print("✅ Saved: maxcrnn_phase4.keras")

# Save label encoder
with open('label_encoder.pkl', 'wb') as f:
    pickle.dump(le, f)
print("✅ Saved: label_encoder.pkl")
```

```
✅ Saved: random_forest_phase4.pkl
✅ Saved: maxcrnn_phase4.keras
✅ Saved: label_encoder.pkl
```

```
# Download results
from google.colab import files
```

```
# Zip everything
!zip -r phase4_results.zip *.pkl *.keras *.png

files.download('phase4_results.zip')
```

```
  adding: label_encoder.pkl (deflated 24%)
  adding: random_forest_phase4.pkl (deflated 80%)
  adding: maxcrnn_phase4.keras (deflated 10%)
  adding: maxcrnn_confusion_matrix.png (deflated 16%)
  adding: maxcrnn_training_curves.png (deflated 7%)
  adding: model_comparison.png (deflated 24%)
  adding: rf_confusion_matrix.png (deflated 15%)
  adding: sanity_check_mouthing.png (deflated 17%)
  adding: viz_adc_distribution.png (deflated 21%)
  adding: viz_amplitude_comparison.png (deflated 13%)
  adding: viz_random_samples_mouthing.png (deflated 4%)
  adding: viz_random_samples_subvocal.png (deflated 5%)
```

📊 Summary

Model	Val Acc (L3)	Test Acc (L4)	Transfer Gap
Random Forest	TBD	TBD	TBD
MaxCRNN	TBD	TBD	TBD

Next Steps:

1. Fill TBD values after running
2. Export confusion matrices for assignment
3. Analyze per-class performance

🚀 ADVANCED STRATEGIES FOR ACCURACY IMPROVEMENT

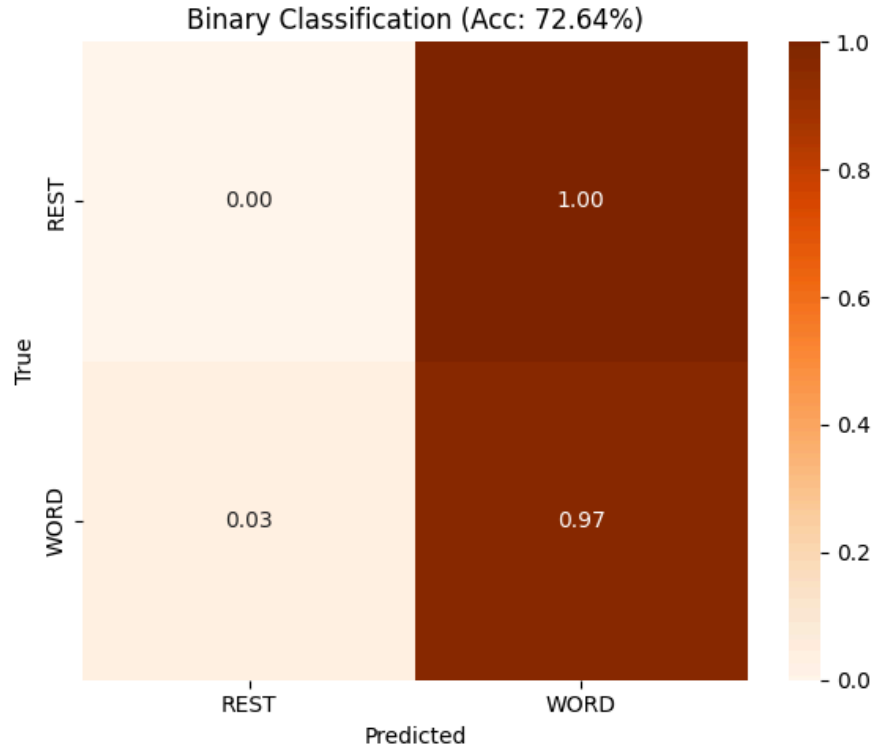
9️⃣ Binary Classification: WORD vs REST

Before attempting 4-class classification, let's see if we can at least distinguish "any word" from "rest". This is often easier and can give us confidence in the signal quality.

```
print("=" * 60)
print("🎯 STRATEGY 1: Binary Classification (WORD vs REST)")
print("=" * 60)

# Convert to binary labels
def to_binary(y, le):
    """Convert multi-class labels to binary (WORD vs REST)."""
    class_names = le.classes_
    rest_idx = np.where(class_names == 'REST')[0][0] if 'REST' in class_names else -1
    return (y != rest_idx).astype(int)  # 0=REST, 1=WORD

y_train_binary = to_binary(y_train, le)
y_val_binary = to_binary(y_val, le)
y_test_binary = to_binary(y_test, le)

print(f"Binary distribution (train): REST={np.sum(y_train_binary==0)}, WORD={np.sum(y_train_binary==1)}")

# Train binary RF
rf_binary = RandomForestClassifier(n_estimators=200, max_depth=20, random_state=1738, n_jobs=-1)
rf_binary.fit(X_train_feat_aug, np.tile(y_train_binary, 3))  # Match augmented size

# Evaluate
y_test_pred_binary = rf_binary.predict(X_test_feat)
binary_acc = accuracy_score(y_test_binary, y_test_pred_binary)
```

```
print(f"\n✅ Binary Accuracy (WORD vs REST): {binary_acc:.4f}")
print(f"   (If this is high, signal is good but words are hard to distinguish)")

# Confusion matrix
plt.figure(figsize=(6, 5))
cm_binary = confusion_matrix(y_test_binary, y_test_pred_binary, normalize='true')
sns.heatmap(cm_binary, annot=True, fmt='.2f', cmap='Oranges',
            xticklabels=['REST', 'WORD'], yticklabels=['REST', 'WORD'])
plt.title(f'Binary Classification (Acc: {binary_acc:.2%})')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.tight_layout()
plt.savefig('binary_confusion_matrix.png', dpi=150)
plt.show()
```

```
============================================================
🎯 STRATEGY 1: Binary Classification (WORD vs REST)
============================================================
Binary distribution (train): REST=42, WORD=128

✅ Binary Accuracy (WORD vs REST): 0.7264
   (If this is high, signal is good but words are hard to distinguish)
```



🔟 Spectrogram + 2D CNN (ImageNet Transfer Learning)

Convert 1D signals to mel-spectrograms and use MobileNetV2 pretrained on ImageNet. This leverages millions of image-trained weights for pattern recognition.

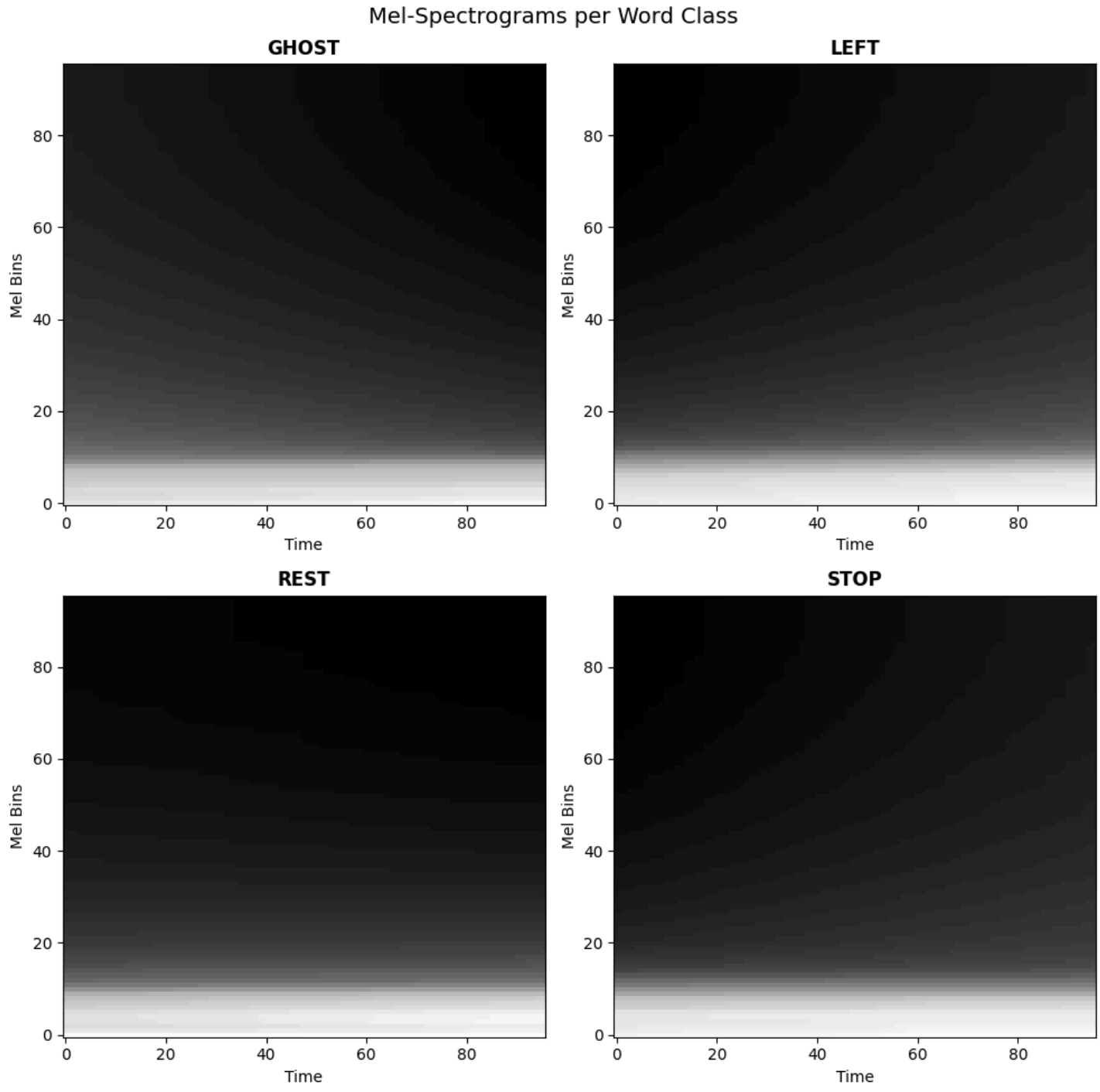
```
print("=" * 60)
print("🎯 STRATEGY 2: Spectrogram + MobileNetV2 Transfer Learning")
print("=" * 60)

import librosa
import librosa.display

def signal_to_spectrogram(signal, sr=1000, n_mels=64, target_size=(96, 96)):
    """
    Convert 1D EMG signal to mel-spectrogram image.
    """
    # Compute mel-spectrogram
    S = librosa.feature.melspectrogram(
```

```
        y=signal.astype(float),
        sr=sr,
        n_mels=n_mels,
        fmax=sr/2
    )
    # Convert to dB
    S_dB = librosa.power_to_db(S, ref=np.max)

    # Normalize to 0-255
    S_norm = ((S_dB - S_dB.min()) / (S_dB.max() - S_dB.min() + 1e-8) * 255).astype(np.uint8)

    # Resize to target size
    from scipy.ndimage import zoom
    zoom_factors = (target_size[0] / S_norm.shape[0], target_size[1] / S_norm.shape[1])
    S_resized = zoom(S_norm, zoom_factors, order=1)

    # Convert to 3-channel (RGB) for ImageNet models
    S_rgb = np.stack([S_resized, S_resized, S_resized], axis=-1)

    return S_rgb

# Create spectrogram dataset
print("\nConverting signals to spectrograms...")
X_train_spec = np.array([signal_to_spectrogram(x.flatten()) for x in X_train])
X_val_spec = np.array([signal_to_spectrogram(x.flatten()) for x in X_val])
X_test_spec = np.array([signal_to_spectrogram(x.flatten()) for x in X_test])

print(f"Spectrogram shapes: Train {X_train_spec.shape}, Val {X_val_spec.shape}, Test {X_test_spec.shape}")
```

```
============================================================
🎯 STRATEGY 2: Spectrogram + MobileNetV2 Transfer Learning
============================================================

Converting signals to spectrograms...
Spectrogram shapes: Train (170, 96, 96, 3), Val (30, 96, 96, 3), Test (201, 96, 96, 3)
```

```
# Visualize sample spectrograms
print("\n📊 Sample spectrograms per class:")
fig, axes = plt.subplots(2, 2, figsize=(10, 10))
axes = axes.flatten()

for i, cls in enumerate(le.classes_):
    cls_idx = np.where(le.classes_ == cls)[0][0]
    sample_indices = np.where(y_train == cls_idx)[0]
    if len(sample_indices) > 0:
        idx = sample_indices[0]
        axes[i].imshow(X_train_spec[idx], aspect='auto', origin='lower')
        axes[i].set_title(f'{cls}', fontsize=12, fontweight='bold')
        axes[i].set_xlabel('Time')
        axes[i].set_ylabel('Mel Bins')

plt.suptitle('Mel-Spectrograms per Word Class', fontsize=14)
plt.tight_layout()
plt.savefig('viz_spectrograms.png', dpi=150)
plt.show()
```

Sample spectrograms per class:




```
# Build MobileNetV2 transfer learning model
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout

def build_spectrogram_cnn(input_shape=(96, 96, 3), n_classes=4):
    """
    MobileNetV2 with ImageNet weights for spectrogram classification.
    """
    # Load pretrained MobileNetV2
    base_model = MobileNetV2(
        weights='imagenet',
        include_top=False,
        input_shape=input_shape
    )

    # Freeze base layers
```

```
    for layer in base_model.layers:
        layer.trainable = False

    # Add classification head
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(128, activation='relu')(x)
    x = Dropout(0.5)(x)
    outputs = Dense(n_classes, activation='softmax')(x)

    model = Model(base_model.input, outputs, name='SpecCNN_MobileNetV2')
    return model

print("\nBuilding MobileNetV2 spectrogram model...")
spec_model = build_spectrogram_cnn(n_classes=len(le.classes_))
spec_model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
print(f"Total params: {spec_model.count_params():,}")
```

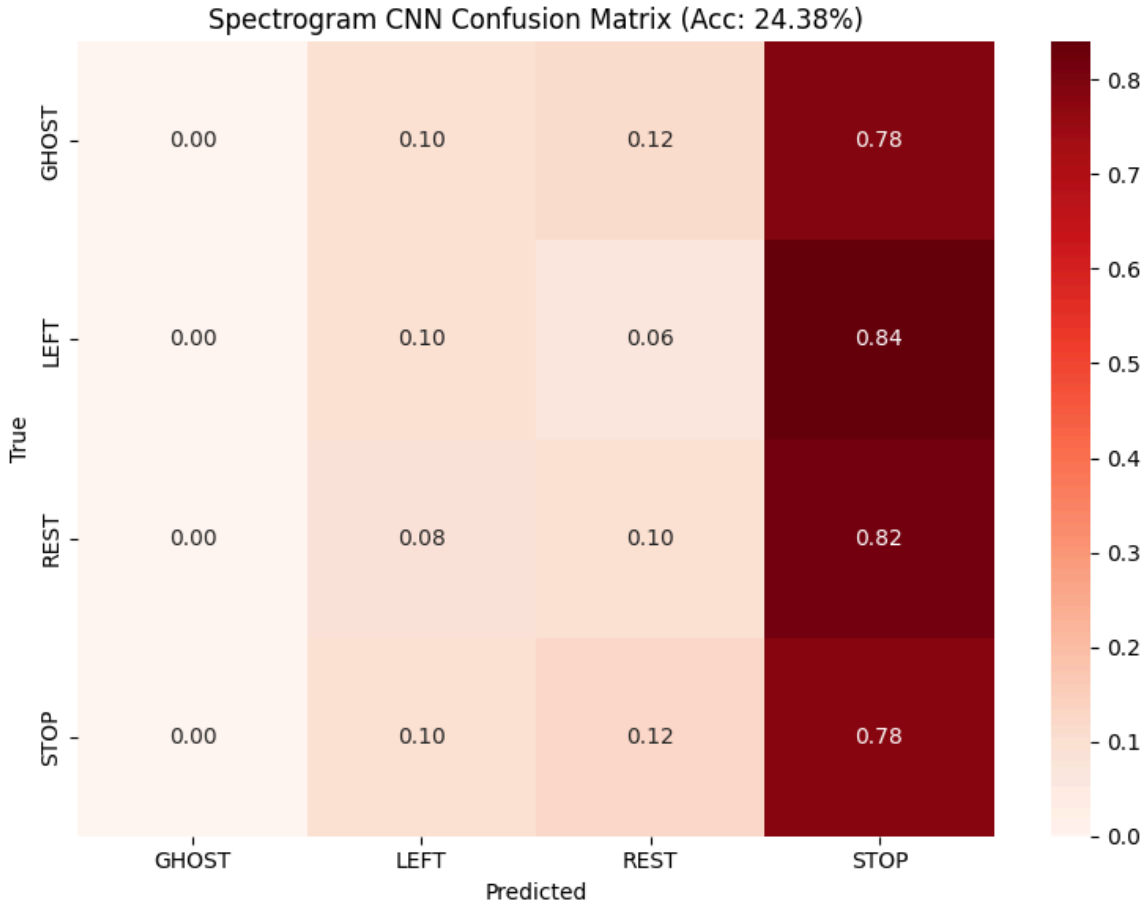
```
Building MobileNetV2 spectrogram model...
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2
9406464/9406464 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Total params: 2,422,468
```

```
# Train spectrogram model
print("\n🚀 Training Spectrogram CNN...")
spec_callbacks = [
    EarlyStopping(monitor='val_loss', patience=15, restore_best_weights=True),
    ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, min_lr=1e-6)
]

# Normalize spectrograms for ImageNet
X_train_spec_norm = X_train_spec / 255.0
X_val_spec_norm = X_val_spec / 255.0
X_test_spec_norm = X_test_spec / 255.0

spec_history = spec_model.fit(
    X_train_spec_norm, y_train,
    validation_data=(X_val_spec_norm, y_val),
    epochs=50,
    batch_size=16,
    callbacks=spec_callbacks,
    verbose=1
)
```

```
11/11 ━━━━━━━━━━━━━━━━━━━━ 5s 437ms/step - accuracy: 0.2310 - loss: 1.3786 - val_accuracy: 0.2667 - val_los
Epoch 34/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 3s 290ms/step - accuracy: 0.2321 - loss: 1.3812 - val_accuracy: 0.2667 - val_los
Epoch 35/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 4s 332ms/step - accuracy: 0.2262 - loss: 1.3781 - val_accuracy: 0.3667 - val_los
Epoch 36/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 4s 311ms/step - accuracy: 0.2625 - loss: 1.3782 - val_accuracy: 0.4000 - val_los
Epoch 37/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 4s 179ms/step - accuracy: 0.3138 - loss: 1.3705 - val_accuracy: 0.3000 - val_los
Epoch 38/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 2s 154ms/step - accuracy: 0.2779 - loss: 1.3735 - val_accuracy: 0.3333 - val_los
Epoch 39/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 2s 176ms/step - accuracy: 0.2622 - loss: 1.3714 - val_accuracy: 0.4000 - val_los
Epoch 40/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 2s 196ms/step - accuracy: 0.2730 - loss: 1.3789 - val_accuracy: 0.3667 - val_los
Epoch 41/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 4s 348ms/step - accuracy: 0.2601 - loss: 1.3724 - val_accuracy: 0.3333 - val_los
Epoch 42/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 2s 152ms/step - accuracy: 0.2893 - loss: 1.3750 - val_accuracy: 0.2667 - val_los
Epoch 43/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 2s 146ms/step - accuracy: 0.2281 - loss: 1.3737 - val_accuracy: 0.2667 - val_los
Epoch 44/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 2s 154ms/step - accuracy: 0.2528 - loss: 1.3816 - val_accuracy: 0.3000 - val_los
Epoch 45/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 2s 152ms/step - accuracy: 0.2978 - loss: 1.3684 - val_accuracy: 0.2667 - val_los
Epoch 46/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 2s 147ms/step - accuracy: 0.2737 - loss: 1.3633 - val_accuracy: 0.2333 - val_los
Epoch 47/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 2s 209ms/step - accuracy: 0.2370 - loss: 1.3820 - val_accuracy: 0.2333 - val_los
Epoch 48/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 2s 217ms/step - accuracy: 0.2568 - loss: 1.3712 - val_accuracy: 0.2667 - val_los
Epoch 49/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 2s 160ms/step - accuracy: 0.2606 - loss: 1.3688 - val_accuracy: 0.3000 - val_los
Epoch 50/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 2s 157ms/step - accuracy: 0.2814 - loss: 1.3738 - val_accuracy: 0.3000 - val_los
```

```
# Evaluate spectrogram model
print("\n📊 Spectrogram CNN Evaluation:")
val_loss, val_acc_spec = spec_model.evaluate(X_val_spec_norm, y_val, verbose=0)
test_loss, test_acc_spec = spec_model.evaluate(X_test_spec_norm, y_test, verbose=0)

print(f"Val Accuracy (L3): {val_acc_spec:.4f}")
print(f"Test Accuracy (L4): {test_acc_spec:.4f}")
print(f"Transfer Gap: {val_acc_spec - test_acc_spec:.4f}")

# Confusion matrix
y_test_pred_spec = np.argmax(spec_model.predict(X_test_spec_norm), axis=1)
plt.figure(figsize=(8, 6))
cm_spec = confusion_matrix(y_test, y_test_pred_spec, normalize='true')
sns.heatmap(cm_spec, annot=True, fmt='.2f', cmap='Reds',
            xticklabels=le.classes_, yticklabels=le.classes_)
plt.title(f'Spectrogram CNN Confusion Matrix (Acc: {test_acc_spec:.2%})')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.tight_layout()
plt.savefig('spectrogram_cnn_confusion.png', dpi=150)
plt.show()
```

```
📊 Spectrogram CNN Evaluation:
Val Accuracy (L3): 0.3000
Test Accuracy (L4): 0.2438
Transfer Gap: 0.0562
7/7 ━━━━━━━━━━━━━━━━━━━━ 5s 543ms/step
```



1️⃣1️⃣ Window Overlap Strategy

Instead of non-overlapping windows, use 50% overlap to create more training samples.

```
print("=" * 60)
print("🎯 STRATEGY 3: Window Overlap (50%)")
print("=" * 60)

def create_windows_overlap(df, window_size=1000, overlap=0.5, center_offset=500):
    """
    Create overlapping windows for more training data.

    Args:
        window_size: Window size in samples
        overlap: Overlap fraction (0.5 = 50%)
        center_offset: Where word starts in block
    """
    step = int(window_size * (1 - overlap))
    windows_X = []
    windows_y = []

    df['label_change'] = df['Label'] != df['Label'].shift(1)
    df['block_id'] = df['label_change'].cumsum()

    for block_id, block in df.groupby('block_id'):
        label = block['Label'].iloc[0]
        signal = block['RawValue'].values

        # Slide window with overlap
```