In [1]:
```
from pathlib import Path
import torch, torchaudio
import numpy as np
import json

PROJECT_ROOT = Path("/Users/maddy/Desktop/PLEP/Project/CS-576-Final-Pr
MODEL_DIR = PROJECT_ROOT / "saved_models"
DATA_DIR = PROJECT_ROOT / "sample_data/speech_commands_v0.02"

print("Using:", PROJECT_ROOT)
```

```
Using: /Users/maddy/Desktop/PLEP/Project/CS-576-Final-Project
```

In [3]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F

# ------------------------------
# CNN MODEL (matches baseline_cnn_kws_vfinal.pt)
# ------------------------------
class CNN_KWS(nn.Module):
    def __init__(self, num_classes=6, flatten_dim=3840):
        super().__init__()

        self.features = nn.Sequential(
            nn.Conv2d(1, 8, kernel_size=5, stride=1, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Conv2d(8, 16, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),
        )

        # IMPORTANT: fixed flatten_dim = 3840 to match checkpoint
        self.classifier = nn.Sequential(
            nn.Linear(flatten_dim, 64),
            nn.ReLU(),
            nn.Linear(64, num_classes),
        )

    def forward(self, x):
        # x: [B,40,T]
        x = x.unsqueeze(1)
        x = self.features(x)
        x = torch.flatten(x, 1)

        # ensure correct shape
        Fdim = x.shape[1]
        if Fdim > 3840:
            x = x[:, :3840]
        elif Fdim < 3840:
```

```
            x = F.pad(x, (0, 3840 - Fdim))

        return self.classifier(x)


# ------------------------------
# SNN MODEL (same from snn_conversion)
# ------------------------------
import snntorch as snn
from snntorch import surrogate

spike_grad = surrogate.fast_sigmoid()

class SNN_KWS(nn.Module):
    def __init__(self, base_cnn: CNN_KWS, num_steps: int = 50, beta: f
        super().__init__()
        self.num_steps = num_steps

        # copy weights
        self.features = base_cnn.features
        self.fc1 = base_cnn.classifier[0]
        self.fc2 = base_cnn.classifier[2]

        # SNN layers
        self.lif1 = snn.Leaky(beta=beta, spike_grad=spike_grad)
        self.lif2 = snn.Leaky(beta=beta, spike_grad=spike_grad)

    def forward(self, x):
        spk2_rec = []
        mem1 = self.lif1.init_leaky()
        mem2 = self.lif2.init_leaky()

        x = x.unsqueeze(1)

        for _ in range(self.num_steps):
            cur = self.features(x)
            cur = torch.flatten(cur, 1)
            cur = F.relu(self.fc1(cur))
            spk1, mem1 = self.lif1(cur, mem1)
            cur2 = self.fc2(spk1)
            spk2, mem2 = self.lif2(cur2, mem2)
            spk2_rec.append(spk2)

        return torch.stack(spk2_rec)  # [T,B,C]


# ------------------------------
# FEATURE EXTRACTION FOR LOIHI
# ------------------------------
def extract_cnn_features(x: torch.Tensor, model: CNN_KWS):
    with torch.no_grad():
        x_ = x.unsqueeze(1)
```

```
        h = model.features(x_)
        h = torch.flatten(h, 1)
        h = F.relu(model.classifier[0](h))
    return h   # [B,64]
```

In [7]:
```
# %%
import torch

# Select CPU or GPU
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)
```

```
Using device: cpu
```

In [8]:
```
# %%
import soundfile as sf
import torchaudio
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader
from pathlib import Path

# ----------------------------------------
# Dataset and MFCC Transform
# ----------------------------------------

CLASSES = ["yes", "no", "go", "stop", "down", "up"]
SAMPLE_RATE = 16000
N_MFCC = 40

mfcc_transform = torchaudio.transforms.MFCC(
    sample_rate=SAMPLE_RATE,
    n_mfcc=N_MFCC,
    melkwargs={
        "n_fft": 400,
        "hop_length": 160,
        "n_mels": 40,
        "center": False,
    },
)

def wav_to_mfcc(path: str) -> torch.Tensor:
    # Load audio with soundfile to avoid torchcodec issues
    waveform, sr = sf.read(path)
    waveform = torch.tensor(waveform).float().unsqueeze(0)

    if sr != SAMPLE_RATE:
        waveform = torchaudio.functional.resample(waveform, sr, SAMPLE

    mfcc = mfcc_transform(waveform).squeeze(0)
    mfcc = (mfcc - mfcc.mean()) / (mfcc.std() + 1e-6)   # normalize
    return mfcc
```

```
# ------------------------------------
# Dataset Class
# ------------------------------------

class KWSDataset(Dataset):
    def __init__(self, files, classes):
        self.files = files
        self.classes = classes

    def __len__(self):
        return len(self.files)

    def __getitem__(self, idx):
        path = self.files[idx]
        mfcc = wav_to_mfcc(str(path))
        label = self.classes.index(path.parent.name)
        return mfcc, label


# ------------------------------------
# Collate (pad MFCCs to equal T)
# ------------------------------------

def pad_collate(batch):
    xs, ys = zip(*batch)
    max_t = max(x.shape[1] for x in xs)
    xs = [F.pad(x, (0, max_t - x.shape[1])) for x in xs]
    return torch.stack(xs), torch.tensor(ys)


# ------------------------------------
# Create Test Loader
# ------------------------------------

file_list = list(DATA_DIR.rglob("*.wav"))
print("Total audio files found:", len(file_list))

test_loader = DataLoader(
    KWSDataset(file_list, CLASSES),
    batch_size=16,
    shuffle=True,
    collate_fn=pad_collate
)

print("Test loader ready.")
```

```
Total audio files found: 105835
Test loader ready.
```

In [9]:

```
# %%
# ------------------------------
# LOAD TRAINED MODELS
# ------------------------------
```

```
cnn_ckpt_path = MODEL_DIR / "baseline_cnn_kws_vfinal.pt"
snn_ckpt_path = MODEL_DIR / "snn_kws_beta0.95_T50.pt"

print("CNN checkpoint exists:", cnn_ckpt_path.exists())
print("SNN checkpoint exists:", snn_ckpt_path.exists())

# --- Load CNN ---
cnn_model = CNN_KWS(num_classes=6, flatten_dim=3840).to(device)
cnn_state = torch.load(cnn_ckpt_path, map_location=device)
cnn_model.load_state_dict(cnn_state)
cnn_model.eval()
print("Loaded CNN model.")

# --- Load SNN ---
snn_model = SNN_KWS(cnn_model, num_steps=50, beta=0.95).to(device)

if snn_ckpt_path.exists():
    snn_state = torch.load(snn_ckpt_path, map_location=device)
    snn_model.load_state_dict(snn_state)
    print("Loaded SNN model.")
else:
    print(" SNN checkpoint missing — using CNN weights only.")

snn_model.eval()
```

```
CNN checkpoint exists: True
SNN checkpoint exists: True
Loaded CNN model.
Loaded SNN model.
```

Out[9]:
```
SNN_KWS(
  (features): Sequential(
    (0): Conv2d(1, 8, kernel_size=(5, 5), stride=(1, 1), padding=(2,
2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ce
il_mode=False)
    (3): Conv2d(8, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ce
il_mode=False)
  )
  (fc1): Linear(in_features=3840, out_features=64, bias=True)
  (fc2): Linear(in_features=64, out_features=6, bias=True)
  (lif1): Leaky()
  (lif2): Leaky()
)
```

In [10]:
```
# %%
def eval_cnn(model, loader):
    model.eval()
    correct = 0
```

```
    total = 0

    with torch.no_grad():
        for x, y in loader:
            x = x.to(device)      # [B,40,T]
            y = y.to(device)

            logits = model(x)     # output [B,6]
            preds = logits.argmax(dim=1)

            correct += (preds == y).sum().item()
            total += y.size(0)

    return correct / max(total, 1)


# ---- Run Evaluation ----
cnn_acc = eval_cnn(cnn_model, test_loader)
print(f"CNN Accuracy: {cnn_acc*100:.2f}%")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[10], line 22
     18     return correct / max(total, 1)
     21 # ---- Run Evaluation ----
---> 22 cnn_acc = eval_cnn(cnn_model, test_loader)
     23 print(f"CNN Accuracy: {cnn_acc*100:.2f}%")

Cell In[10], line 8, in eval_cnn(model, loader)
      5 total = 0
      7 with torch.no_grad():
----> 8     for x, y in loader:
      9         x = x.to(device)      # [B,40,T]
     10         y = y.to(device)

File /opt/miniconda3/envs/cs576/lib/python3.10/site-packages/torch/utils/data/dataloader.py:732, in _BaseDataLoaderIter.__next__(self)
    729 if self._sampler_iter is None:
    730     # TODO(https://github.com/pytorch/pytorch/issues/76750)
    731     self._reset()  # type: ignore[call-arg]
--> 732 data = self._next_data()
    733 self._num_yielded += 1
    734 if (
    735     self._dataset_kind == _DatasetKind.Iterable
    736     and self._IterableDataset_len_called is not None
    737     and self._num_yielded > self._IterableDataset_len_called
    738 ):

File /opt/miniconda3/envs/cs576/lib/python3.10/site-packages/torch/utils/data/dataloader.py:788, in _SingleProcessDataLoaderIter._next_data(self)
```

```
    786 def _next_data(self):
    787     index = self._next_index()  # may raise StopIteration
--> 788     data = self._dataset_fetcher.fetch(index)  # may raise Stop
Iteration
    789     if self._pin_memory:
    790         data = _utils.pin_memory.pin_memory(data, self._pin_mem
ory_device)

File /opt/miniconda3/envs/cs576/lib/python3.10/site-packages/torch/util
s/data/_utils/fetch.py:52, in _MapDatasetFetcher.fetch(self, possibly_b
atched_index)
     50         data = self.dataset.__getitems__(possibly_batched_inde
x)
     51     else:
---> 52         data = [self.dataset[idx] for idx in possibly_batched_i
ndex]
     53 else:
     54     data = self.dataset[possibly_batched_index]

File /opt/miniconda3/envs/cs576/lib/python3.10/site-packages/torch/util
s/data/_utils/fetch.py:52, in <listcomp>(.0)
     50         data = self.dataset.__getitems__(possibly_batched_inde
x)
     51     else:
---> 52         data = [self.dataset[idx] for idx in possibly_batched_i
ndex]
     53 else:
     54     data = self.dataset[possibly_batched_index]

Cell In[8], line 55, in KWSDataset.__getitem__(self, idx)
     53 path = self.files[idx]
     54 mfcc = wav_to_mfcc(str(path))
---> 55 label = self.classes.index(path.parent.name)
     56 return mfcc, label

ValueError: 'seven' is not in list
```

In [ ]: