BitePulse AI - Modeling: RGB 3D CNN vs Hyperband 3D CNN

- **Name/Group:** Aktham Almomani / Group 13

Introduction

In this notebook we work with RGB clips instead of pose:

- Load window level labels and frame index for the EatSense videos.
- Build a **PyTorch 3D CNN baseline** that takes short RGB clips and predicts intake vs non intake.
- Build a **Keras 3D CNN** and use **Keras Tuner Hyperband** to search over architecture and training hyperparameters.
- Evaluate both models with exactly the same **window level metrics and plots** (confusion matrix, ROC, PR), so we can compare them and with the pose TCN.

```
# Mount Google Drive:
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
!pip install -q keras-tuner
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0.0/129.4 kB ? eta -:--:--
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 129.4/129.4 kB 4.2 MB/s eta
0:00:00
```

```
import os, sys, math, json, time, random, logging, re
from pathlib import Path
from typing import Dict, Any, Optional

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from PIL import Image
import subprocess, shutil

import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader,
WeightedRandomSampler

import tensorflow as tf
import keras_tuner as kt
```

```python
from sklearn.metrics import (
    precision_recall_curve, roc_curve, roc_auc_score,
average_precision_score,
    accuracy_score, precision_score, recall_score, f1_score,
    balanced_accuracy_score, matthews_corrcoef, confusion_matrix, auc,
    classification_report
)

from tqdm.auto import tqdm
from contextlib import nullcontext

plt.rcParams["figure.dpi"] = 120

POSE_FEATS_PATH =
Path("/content/drive/MyDrive/eatsense/windows/pose_feats.parquet")

pose_feats = pd.read_parquet(POSE_FEATS_PATH)
print("pose_feats shape:", pose_feats.shape)
pose_feats.head()
```

```
pose_feats shape: (98582, 18)
```


```
{"summary":"{\n  \"name\": \"pose_feats\",\n  \"rows\": 98582,\n  
\"fields\": [\n    {\n      \"column\": \"key\",\n      
\"properties\": {\n        \"dtype\": \"category\",\n        
\"num_unique_values\": 135,\n        \"samples\": [\n          
\"20230110_114618\",\n          \"20221013_131254\",\n          
\"20230110_134822\"\n        ],\n        \"semantic_type\": \"\",\n        
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": 
\"split\",\n      \"properties\": {\n        \"dtype\": \"category\",\
n        \"num_unique_values\": 3,\n        \"samples\": [\n          
\"train\",\n          \"val\",\n          \"test\"\n        ],\n        
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"win_id\",\n      \"properties\": 
{\n        \"dtype\": \"number\",\n        \"std\": 388,\n        
\"min\": 0,\n        \"max\": 2072,\n        \"num_unique_values\": 
2073,\n        \"samples\": [\n          1498,\n          1333,\n          
1808\n        ],\n        \"semantic_type\": \"\",\n        
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": 
\"start_sec\",\n      \"properties\": {\n        \"dtype\": 
\"number\",\n        \"std\": 194.03245452862419,\n        \"min\": 
0.0,\n        \"max\": 1036.0,\n        \"num_unique_values\": 2073,\n        
\"samples\": [\n          749.0,\n          666.5,\n          904.0\n        
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      
}\n    },\n    {\n      \"column\": \"end_sec\",\n      
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 
194.03245452862419,\n        \"min\": 2.0,\n        \"max\": 1038.0,\n        
\"num_unique_values\": 2073,\n        \"samples\": [\n          
751.0,\n          668.5,\n          906.0\n        ],\n
```

```
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"label\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 0,\n        \"min\":
0,\n        \"max\": 1,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          1,\n          0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"rw_speed_mean\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
24.33982188285745,\n        \"min\": 0.00801266320426811,\n
\"max\": 1105.2935333031576,\n        \"num_unique_values\": 98582,\n
\"samples\": [\n          30.13531184750308,\n
22.075381546873665\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"rw_speed_max\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 110.32374073704662,\n        \"min\":
0.0222050772136882 2,\n        \"max\": 6614.9512166947,\n
\"num_unique_values\": 57853,\n        \"samples\": [\n
9.455119186054414,\n          10.018440956556741\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"rw_head_min\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
39.09028959178826,\n        \"min\": 0.5873013515401203,\n
\"max\": 271.81711304963875,\n        \"num_unique_values\": 57975,\n
\"samples\": [\n          118.0655045994019,\n
52.37232298782691\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"rw_path\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 51.25176255611852,\n        \"min\":
0.01940676395299922,\n        \"max\": 2127.9823211899215,\n
\"num_unique_values\": 98582,\n        \"samples\": [\n
59.134021145496476,\n          48.0751711089649\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"lw_speed_mean\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
24.452837158993937,\n        \"min\": 5.573618882666242e-13,\n
\"max\": 928.8229979693119,\n        \"num_unique_values\": 98582,\n
\"samples\": [\n          18.55339536937793,\n
1.830677739083311\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"lw_speed_max\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 122.74647351784876,\n        \"min\":
6.431098710768743e-13,\n        \"max\": 6604.368538089087,\n
\"num_unique_values\": 59144,\n        \"samples\": [\n
63.792844095754475,\n          12.904262946985533\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"lw_head_min\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
41.098975018434494,\n        \"min\": 0.0,\n        \"max\":
294.5214109513494,\n        \"num_unique_values\": 59372,\n
```

```
\"samples\": [\n          103.79806326896166,\n          169.5701838712913\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"lw_path\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 58.380739269269306,\n        \"min\": 0.0,\n        \"max\": 2479.686875105099,\n        \"num_unique_values\": 98575,\n        \"samples\": [\n          18.687669860059938,\n          10.369406213105538\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"re_angle_mean\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 41.810937801640385,\n        \"min\": 0.3569787476870709,\n        \"max\": 179.8201187446251,\n        \"num_unique_values\": 98582,\n        \"samples\": [\n          53.94185320087705,\n          63.571043576705115\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"re_angle_std\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 15.313709858756834,\n        \"min\": 0.006132719445273099,\n        \"max\": 87.05938228479292,\n        \"num_unique_values\": 98582,\n        \"samples\": [\n          11.687669535127446,\n          16.99852090849505\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"le_angle_mean\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 42.150906378037746,\n        \"min\": 0.13646436532583334,\n        \"max\": 179.90475920540828,\n        \"num_unique_values\": 98582,\n        \"samples\": [\n          60.94813611973014,\n          98.9314811702009\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"le_angle_std\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 11.510725091381751,\n        \"min\": 0.010677667435140608,\n        \"max\": 84.96826767832528,\n        \"num_unique_values\": 98582,\n        \"samples\": [\n          21.94339713867533,\n          3.5710967735075645\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe","variable_name":"pose_feats"}
```


```
cols_needed = ["key", "start_sec", "end_sec", "label", "split"]

windows_df = (
    pose_feats[cols_needed]
    .drop_duplicates()
    .reset_index(drop=True)
)

print("windows_df shape:", windows_df.shape)

windows_df["split"].value_counts()
```

```
windows_df shape: (98582, 5)
```

```
split
train    62568
val      21184
test     14830
Name: count, dtype: int64
```

```
windows_df.head()
```


```
{"summary":"{\n  \"name\": \"windows_df\",\n  \"rows\": 98582,\n
\"fields\": [\n    {\n      \"column\": \"key\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 135,\n        \"samples\": [\n
\"20230110_114618\",\n          \"20221013_131254\",\n
\"20230110_134822\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"start_sec\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 194.03245452862419,\n        \"min\":
0.0,\n        \"max\": 1036.0,\n        \"num_unique_values\": 2073,\n
\"samples\": [\n          749.0,\n          666.5,\n          904.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"end_sec\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
194.03245452862419,\n        \"min\": 2.0,\n        \"max\": 1038.0,\n
\"num_unique_values\": 2073,\n        \"samples\": [\n
751.0,\n          668.5,\n          906.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"label\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 0,\n        \"min\":
0,\n        \"max\": 1,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          1,\n          0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"split\",\n      \"properties\": {\
n        \"dtype\": \"category\",\n        \"num_unique_values\": 3,\n
\"samples\": [\n          \"train\",\n          \"val\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"windows_df"}
```


Extract RGB Frames to Disk

Scan the EatSense RGB videos, match each labeled `key` to its source clip, and use `ffmpeg` to extract center-cropped 224x224 PNG frames at 15 FPS into `FRAMES_DIR` for the train/val/test splits.

```
# Path:
VIDEO_ROOTS = [
    Path("/content/drive/MyDrive/eatsense/rgb/deepfaked")

]
```

```python
FRAMES_DIR = Path("/content/drive/MyDrive/eatsense/frames")
FRAMES_DIR.mkdir(parents=True, exist_ok=True)

# Splits to process:
SPLITS_TO_PROCESS = {"train", "val", "test"}

# Frame extraction settings:
FPS_TARGET = 15
TARGET_W, TARGET_H = 224, 224  # standard size for 3D-CNN inputs:
VIDEO_EXTS = {".mp4", ".mov", ".mkv", ".avi", ".m4v"}

split_col = windows_df["split"].astype(str).str.lower()
keys_needed = (
    windows_df.loc[split_col.isin(SPLITS_TO_PROCESS), "key"]
    .dropna().astype(str).unique().tolist()
)
print(f"Keys to process: {len(keys_needed)}")

# Index available videos once:
def all_videos_under(roots):
    out = []
    for r in roots:
        if not r.exists():
            continue
        for p in r.rglob("*"):
            if p.is_file() and p.suffix.lower() in VIDEO_EXTS:
                out.append(p)
    return out

videos = all_videos_under(VIDEO_ROOTS)
print(f"Found videos: {len(videos)}")
print("Sample:", [str(v) for v in videos[:3]])

# Heuristic matcher: map each key -> video path:
def match_video_for_key(k: str):
    k_low = k.lower()
    # exact stem match:
    for v in videos:
        if v.stem.lower() == k_low:
            return v
    # relaxed match (containment / prefix):
    for v in videos:
        stem = v.stem.lower()
        if (k_low in stem) or (stem in k_low) or
stem.startswith(k_low) or k_low.startswith(stem):
            return v
    return None

key2vid = {k: match_video_for_key(k) for k in keys_needed}
missing = [k for k, v in key2vid.items() if v is None]
```

```python
print(f"Matched: {sum(v is not None for v in key2vid.values())} |
Missing: {len(missing)}")
if missing:
    print("Missing examples (first 10):", missing[:10])

# Extract frames with ffmpeg (fps + center-crop square + resize to
TARGET_WxTARGET_H):
def extract_frames(video_path: Path, out_dir: Path, fps: int | None =
FPS_TARGET):
    out_dir.mkdir(parents=True, exist_ok=True)
    # Skip if already extracted (has at least one image)
    if any(out_dir.iterdir()):
        return "skip"

    vf = []
    if fps is not None:
        vf.append(f"fps={fps}")
    # center-crop shortest side to square, then high-quality resize:
    vf.append("crop='min(in_w,in_h)':'min(in_w,in_h)'")
    vf.append(f"scale={TARGET_W}:{TARGET_H}:flags=lanczos")

    cmd = [
        "ffmpeg", "-y",
        "-i", str(video_path),
        "-loglevel", "error",
        "-vf", ",".join(vf),
        str(out_dir / "%06d.png"),
    ]
    subprocess.run(cmd, check=True)
    return "ok"

ok, skipped, miss = 0, 0, 0
for k, vid in key2vid.items():
    if vid is None:
        miss += 1
        continue
    out = FRAMES_DIR / k
    try:
        status = extract_frames(vid, out, FPS_TARGET)
        ok += (status == "ok")
        skipped += (status == "skip")
        if status == "ok":
            cnt = sum(1 for _ in out.iterdir())
            print(f"[new] {k}: {cnt} frames -> {out}")
    except Exception as e:
        print(f"[error] {k} <- {vid} :: {e}")

print(f"\nSummary -> new: {ok}, skipped(already there): {skipped},
missing video: {miss}")
print("Frames root:", FRAMES_DIR)
```

```
Keys to process: 135
Found videos: 135
Sample:
['/content/drive/MyDrive/eatsense/rgb/deepfaked/20210518_230219_anonym
ized.mp4',
'/content/drive/MyDrive/eatsense/rgb/deepfaked/20210523_202300_anonymi
zed.mp4',
'/content/drive/MyDrive/eatsense/rgb/deepfaked/20210529_150552_anonymi
zed.mp4']
Matched: 135 | Missing: 0
[new] 20210518_230219: 3668 frames ->
/content/drive/MyDrive/eatsense/frames/20210518_230219
[new] 20210523_202300: 5405 frames ->
/content/drive/MyDrive/eatsense/frames/20210523_202300
[new] 20210529_150552: 7502 frames ->
/content/drive/MyDrive/eatsense/frames/20210529_150552
[new] 20210529_153708: 1397 frames ->
/content/drive/MyDrive/eatsense/frames/20210529_153708
[new] 20210530_153343: 5084 frames ->
/content/drive/MyDrive/eatsense/frames/20210530_153343
[new] 20210531_150448: 5475 frames ->
/content/drive/MyDrive/eatsense/frames/20210531_150448
[new] 20210603_130948: 7740 frames ->
/content/drive/MyDrive/eatsense/frames/20210603_130948
[new] 20210605_155355: 3675 frames ->
/content/drive/MyDrive/eatsense/frames/20210605_155355
[new] 20210606_154234: 4892 frames ->
/content/drive/MyDrive/eatsense/frames/20210606_154234
[new] 20210607_152559: 4247 frames ->
/content/drive/MyDrive/eatsense/frames/20210607_152559
[new] 20210607_154140: 2577 frames ->
/content/drive/MyDrive/eatsense/frames/20210607_154140
[new] 20210608_163819: 7307 frames ->
/content/drive/MyDrive/eatsense/frames/20210608_163819
[new] 20210609_154241: 6053 frames ->
/content/drive/MyDrive/eatsense/frames/20210609_154241
[new] 20210609_215756: 5873 frames ->
/content/drive/MyDrive/eatsense/frames/20210609_215756
[new] 20210609_220502: 1142 frames ->
/content/drive/MyDrive/eatsense/frames/20210609_220502
[new] 20210609_221133: 1232 frames ->
/content/drive/MyDrive/eatsense/frames/20210609_221133
[new] 20210610_144155: 5942 frames ->
/content/drive/MyDrive/eatsense/frames/20210610_144155
[new] 20210610_230110: 8582 frames ->
/content/drive/MyDrive/eatsense/frames/20210610_230110
[new] 20210616_163847: 4382 frames ->
/content/drive/MyDrive/eatsense/frames/20210616_163847
[new] 20210616_170258: 3221 frames ->
/content/drive/MyDrive/eatsense/frames/20210616_170258
```

```
[new] 20210617_165755: 6572 frames ->
/content/drive/MyDrive/eatsense/frames/20210617_165755
[new] 20210617_224119: 5445 frames ->
/content/drive/MyDrive/eatsense/frames/20210617_224119
[new] 20210620_150313: 8550 frames ->
/content/drive/MyDrive/eatsense/frames/20210620_150313
[new] 20210620_230306: 2222 frames ->
/content/drive/MyDrive/eatsense/frames/20210620_230306
[new] 20210620_230707: 4112 frames ->
/content/drive/MyDrive/eatsense/frames/20210620_230707
[new] 20210621_125816: 4688 frames ->
/content/drive/MyDrive/eatsense/frames/20210621_125816
[new] 20210627_142532: 7592 frames ->
/content/drive/MyDrive/eatsense/frames/20210627_142532
[new] 20210714_125156: 6452 frames ->
/content/drive/MyDrive/eatsense/frames/20210714_125156
[new] 20210716_151219: 7352 frames ->
/content/drive/MyDrive/eatsense/frames/20210716_151219
[new] 20210719_135251: 6497 frames ->
/content/drive/MyDrive/eatsense/frames/20210719_135251
[new] 20210728_144106: 9977 frames ->
/content/drive/MyDrive/eatsense/frames/20210728_144106
[new] 20210804_144735: 7802 frames ->
/content/drive/MyDrive/eatsense/frames/20210804_144735
[new] 20210804_145925: 5357 frames ->
/content/drive/MyDrive/eatsense/frames/20210804_145925
[new] 20210810_163907: 3632 frames ->
/content/drive/MyDrive/eatsense/frames/20210810_163907
[new] 20210811_130812: 10247 frames ->
/content/drive/MyDrive/eatsense/frames/20210811_130812
[new] 20210811_132135: 8066 frames ->
/content/drive/MyDrive/eatsense/frames/20210811_132135
[new] 20210813_195607: 3332 frames ->
/content/drive/MyDrive/eatsense/frames/20210813_195607
[new] 20210814_210449: 6452 frames ->
/content/drive/MyDrive/eatsense/frames/20210814_210449
[new] 20210816_191136: 9362 frames ->
/content/drive/MyDrive/eatsense/frames/20210816_191136
[new] 20210923_155848: 10085 frames ->
/content/drive/MyDrive/eatsense/frames/20210923_155848
[new] 20210923_161107: 12819 frames ->
/content/drive/MyDrive/eatsense/frames/20210923_161107
[new] 20210923_163035: 5939 frames ->
/content/drive/MyDrive/eatsense/frames/20210923_163035
[new] 20210923_163801: 6455 frames ->
/content/drive/MyDrive/eatsense/frames/20210923_163801
[new] 20211001_152746: 12447 frames ->
/content/drive/MyDrive/eatsense/frames/20211001_152746
[new] 20211001_154310: 14249 frames ->
/content/drive/MyDrive/eatsense/frames/20211001_154310
```

```
[new] 20211029_144948: 6458 frames ->
/content/drive/MyDrive/eatsense/frames/20211029_144948
[new] 20211029_150526: 2675 frames ->
/content/drive/MyDrive/eatsense/frames/20211029_150526
[new] 20220809_125236: 14180 frames ->
/content/drive/MyDrive/eatsense/frames/20220809_125236
[new] 20220809_142839: 7355 frames ->
/content/drive/MyDrive/eatsense/frames/20220809_142839
[new] 20220809_144052: 7227 frames ->
/content/drive/MyDrive/eatsense/frames/20220809_144052
[new] 20220811_113712: 5078 frames ->
/content/drive/MyDrive/eatsense/frames/20220811_113712
[new] 20220811_114634: 4964 frames ->
/content/drive/MyDrive/eatsense/frames/20220811_114634
[new] 20220811_121550: 4532 frames ->
/content/drive/MyDrive/eatsense/frames/20220811_121550
[new] 20220811_122341: 9792 frames ->
/content/drive/MyDrive/eatsense/frames/20220811_122341
[new] 20220811_124226: 15578 frames ->
/content/drive/MyDrive/eatsense/frames/20220811_124226
[new] 20220812_112034: 2951 frames ->
/content/drive/MyDrive/eatsense/frames/20220812_112034
[new] 20220812_113020: 6797 frames ->
/content/drive/MyDrive/eatsense/frames/20220812_113020
[new] 20220812_115050: 6797 frames ->
/content/drive/MyDrive/eatsense/frames/20220812_115050
[new] 20220812_120241: 7793 frames ->
/content/drive/MyDrive/eatsense/frames/20220812_120241
[new] 20220812_121833: 4562 frames ->
/content/drive/MyDrive/eatsense/frames/20220812_121833
[new] 20220812_122535: 5457 frames ->
/content/drive/MyDrive/eatsense/frames/20220812_122535
[new] 20220812_123723: 2438 frames ->
/content/drive/MyDrive/eatsense/frames/20220812_123723
[new] 20220812_124345: 4676 frames ->
/content/drive/MyDrive/eatsense/frames/20220812_124345
[new] 20220812_125132: 7946 frames ->
/content/drive/MyDrive/eatsense/frames/20220812_125132
[new] 20220812_141747: 11778 frames ->
/content/drive/MyDrive/eatsense/frames/20220812_141747
[new] 20221013_124150: 7401 frames ->
/content/drive/MyDrive/eatsense/frames/20221013_124150
[new] 20221013_125118: 9285 frames ->
/content/drive/MyDrive/eatsense/frames/20221013_125118
[new] 20221013_131254: 9122 frames ->
/content/drive/MyDrive/eatsense/frames/20221013_131254
[new] 20221013_132443: 9567 frames ->
/content/drive/MyDrive/eatsense/frames/20221013_132443
[new] 20221013_134216: 3713 frames ->
/content/drive/MyDrive/eatsense/frames/20221013_134216
```

```
[new] 20221013_135010: 5693 frames ->
/content/drive/MyDrive/eatsense/frames/20221013_135010
[new] 20221013_142709: 3020 frames ->
/content/drive/MyDrive/eatsense/frames/20221013_142709
[new] 20221013_143133: 1862 frames ->
/content/drive/MyDrive/eatsense/frames/20221013_143133
[new] 20221014_113751: 8606 frames ->
/content/drive/MyDrive/eatsense/frames/20221014_113751
[new] 20221014_114952: 5195 frames ->
/content/drive/MyDrive/eatsense/frames/20221014_114952
[new] 20221014_120757: 12842 frames ->
/content/drive/MyDrive/eatsense/frames/20221014_120757
[new] 20221014_122331: 12537 frames ->
/content/drive/MyDrive/eatsense/frames/20221014_122331
[new] 20221014_124303: 9081 frames ->
/content/drive/MyDrive/eatsense/frames/20221014_124303
[new] 20221014_125502: 14012 frames ->
/content/drive/MyDrive/eatsense/frames/20221014_125502
[new] 20221014_140942: 5082 frames ->
/content/drive/MyDrive/eatsense/frames/20221014_140942
[new] 20221014_141711: 9885 frames ->
/content/drive/MyDrive/eatsense/frames/20221014_141711
[new] 20221014_143649: 3719 frames ->
/content/drive/MyDrive/eatsense/frames/20221014_143649
[new] 20221014_144233: 9524 frames ->
/content/drive/MyDrive/eatsense/frames/20221014_144233
[new] 20230109_114530: 3807 frames ->
/content/drive/MyDrive/eatsense/frames/20230109_114530
[new] 20230109_115040: 4067 frames ->
/content/drive/MyDrive/eatsense/frames/20230109_115040
[new] 20230109_115716: 4010 frames ->
/content/drive/MyDrive/eatsense/frames/20230109_115716
[new] 20230109_120255: 2756 frames ->
/content/drive/MyDrive/eatsense/frames/20230109_120255
[new] 20230109_123622: 2634 frames ->
/content/drive/MyDrive/eatsense/frames/20230109_123622
[new] 20230109_124022: 4347 frames ->
/content/drive/MyDrive/eatsense/frames/20230109_124022
[new] 20230109_124729: 3015 frames ->
/content/drive/MyDrive/eatsense/frames/20230109_124729
[new] 20230109_125206: 1452 frames ->
/content/drive/MyDrive/eatsense/frames/20230109_125206
[new] 20230109_132632: 3518 frames ->
/content/drive/MyDrive/eatsense/frames/20230109_132632
[new] 20230109_133112: 3342 frames ->
/content/drive/MyDrive/eatsense/frames/20230109_133112
[new] 20230109_133609: 3630 frames ->
/content/drive/MyDrive/eatsense/frames/20230109_133609
[new] 20230109_134123: 3831 frames ->
/content/drive/MyDrive/eatsense/frames/20230109_134123
```

```
[new] 20230110_113650: 1547 frames ->
/content/drive/MyDrive/eatsense/frames/20230110_113650
[new] 20230110_113938: 1938 frames ->
/content/drive/MyDrive/eatsense/frames/20230110_113938
[new] 20230110_114318: 1748 frames ->
/content/drive/MyDrive/eatsense/frames/20230110_114318
[new] 20230110_114618: 2303 frames ->
/content/drive/MyDrive/eatsense/frames/20230110_114618
[new] 20230110_122933: 2673 frames ->
/content/drive/MyDrive/eatsense/frames/20230110_122933
[new] 20230110_123332: 2183 frames ->
/content/drive/MyDrive/eatsense/frames/20230110_123332
[new] 20230110_123733: 2720 frames ->
/content/drive/MyDrive/eatsense/frames/20230110_123733
[new] 20230110_124143: 2516 frames ->
/content/drive/MyDrive/eatsense/frames/20230110_124143
[new] 20230110_133309: 4886 frames ->
/content/drive/MyDrive/eatsense/frames/20230110_133309
[new] 20230110_133944: 6257 frames ->
/content/drive/MyDrive/eatsense/frames/20230110_133944
[new] 20230110_134822: 4220 frames ->
/content/drive/MyDrive/eatsense/frames/20230110_134822
[new] 20230110_135409: 5303 frames ->
/content/drive/MyDrive/eatsense/frames/20230110_135409
[new] 20230111_114057: 2486 frames ->
/content/drive/MyDrive/eatsense/frames/20230111_114057
[new] 20230111_114640: 2309 frames ->
/content/drive/MyDrive/eatsense/frames/20230111_114640
[new] 20230111_115239: 2817 frames ->
/content/drive/MyDrive/eatsense/frames/20230111_115239
[new] 20230111_115909: 2277 frames ->
/content/drive/MyDrive/eatsense/frames/20230111_115909
[new] 20230124_121223: 3137 frames ->
/content/drive/MyDrive/eatsense/frames/20230124_121223
[new] 20230124_121646: 2793 frames ->
/content/drive/MyDrive/eatsense/frames/20230124_121646
[new] 20230124_122123: 4197 frames ->
/content/drive/MyDrive/eatsense/frames/20230124_122123
[new] 20230124_122705: 2825 frames ->
/content/drive/MyDrive/eatsense/frames/20230124_122705
[new] 20230124_130607: 3989 frames ->
/content/drive/MyDrive/eatsense/frames/20230124_130607
[new] 20230124_131109: 4712 frames ->
/content/drive/MyDrive/eatsense/frames/20230124_131109
[new] 20230124_131740: 4110 frames ->
/content/drive/MyDrive/eatsense/frames/20230124_131740
[new] 20230124_132312: 3981 frames ->
/content/drive/MyDrive/eatsense/frames/20230124_132312
[new] 20230124_135539: 2862 frames ->
/content/drive/MyDrive/eatsense/frames/20230124_135539
```

```
[new] 20230124_135936: 3257 frames ->
/content/drive/MyDrive/eatsense/frames/20230124_135936
[new] 20230124_140432: 5501 frames ->
/content/drive/MyDrive/eatsense/frames/20230124_140432
[new] 20230124_141139: 3078 frames ->
/content/drive/MyDrive/eatsense/frames/20230124_141139
[new] 20230125_140902: 4277 frames ->
/content/drive/MyDrive/eatsense/frames/20230125_140902
[new] 20230125_141519: 3819 frames ->
/content/drive/MyDrive/eatsense/frames/20230125_141519
[new] 20230125_142103: 3722 frames ->
/content/drive/MyDrive/eatsense/frames/20230125_142103
[new] 20230125_142618: 2633 frames ->
/content/drive/MyDrive/eatsense/frames/20230125_142618
[new] 20230127_120259: 2582 frames ->
/content/drive/MyDrive/eatsense/frames/20230127_120259
[new] 20230127_120640: 3561 frames ->
/content/drive/MyDrive/eatsense/frames/20230127_120640
[new] 20230127_121212: 4748 frames ->
/content/drive/MyDrive/eatsense/frames/20230127_121212
[new] 20230127_121834: 3458 frames ->
/content/drive/MyDrive/eatsense/frames/20230127_121834
[new] 20230131_130800: 3626 frames ->
/content/drive/MyDrive/eatsense/frames/20230131_130800
[new] 20230131_131313: 4616 frames ->
/content/drive/MyDrive/eatsense/frames/20230131_131313
[new] 20230131_131949: 6122 frames ->
/content/drive/MyDrive/eatsense/frames/20230131_131949
[new] 20230131_132739: 7133 frames ->
/content/drive/MyDrive/eatsense/frames/20230131_132739

Summary -> new: 135, skipped(already there): 0, missing video: 0
Frames root: /content/drive/MyDrive/eatsense/frames
```

```
test_path =
Path("/content/drive/MyDrive/eatsense/frames/20210816_191136/005820.pn
g")
print(test_path, "exists:", test_path.exists())
```

```
/content/drive/MyDrive/eatsense/frames/20210816_191136/005820.png
exists: True
```

```
FRAMES_DIR = Path("/content/drive/MyDrive/eatsense/frames")

dirs = [p for p in FRAMES_DIR.iterdir() if p.is_dir()]
print("Folders in FRAMES_DIR:", len(dirs))

keys_on_disk   = {d.name for d in dirs}
keys_in_labels = set(keys_needed)
```

```python
print("Missing on disk:",  keys_in_labels - keys_on_disk)
print("Extra on disk:",    keys_on_disk - keys_in_labels)
```

```
Folders in FRAMES_DIR: 135
Missing on disk: set()
Extra on disk: set()
```

```python
rows = []
for d in sorted(dirs, key=lambda p: p.name):
    n_png = sum(1 for _ in d.glob("*.png"))
    rows.append({"key": d.name, "n_frames": n_png})

frames_summary = pd.DataFrame(rows)
frames_summary.head()
```

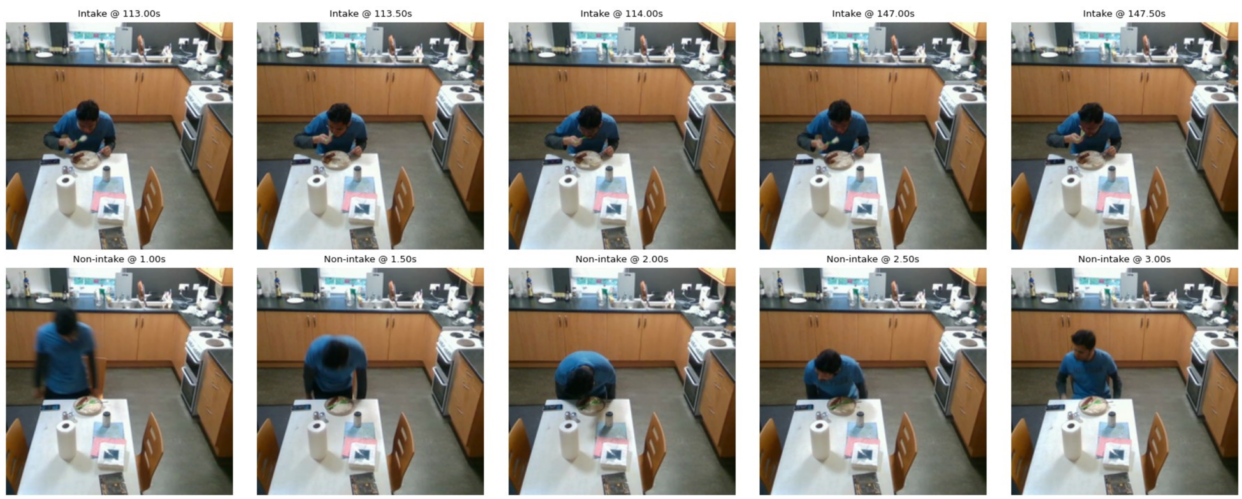
{"summary":"{\n \"name\": \"frames_summary\",\n \"rows\": 135,\n \"fields\": [\n   {\n   \"column\": \"key\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 135,\n   \"samples\": [\n   \"20230110_114618\",\n   \"20221013_131254\",\n   \"20230110_134822\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"n_frames\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 3090,\n   \"min\": 1142,\n   \"max\": 15578,\n   \"num_unique_values\": 133,\n   \"samples\": [\n   4532,\n   3020,\n   7802\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"frames_summary"}

```python
# Now let's look at few examples to confirm the extrcated frames:
CLIP_FPS = 15.0

# Find keys that have BOTH intake (1) and non-intake (0) windows:
keys_with_pos = set(windows_df.loc[windows_df["label"] == 1,
"key"].astype(str))
keys_with_neg = set(windows_df.loc[windows_df["label"] == 0,
"key"].astype(str))
candidate_keys = sorted(list(keys_with_pos & keys_with_neg))

print("Candidate keys with both classes:", len(candidate_keys))
if not candidate_keys:
    raise RuntimeError("No key has both intake and non-intake
windows.")

key = candidate_keys[0]
print("Using key:", key)

# Helper
def frame_path_for_time(key: str, t_sec: float, fps: float =
CLIP_FPS):
    kdir = FRAMES_DIR / str(key)
```

```python
    files = sorted(kdir.glob("*.png"))
    if not files:
        raise RuntimeError(f"No PNG frames for key={key} in {kdir}")
    n = len(files)
    idx = int(round(t_sec * fps))
    idx = max(0, min(n - 1, idx))
    return files[idx]

# Filter windows for this key and select 5 intake + 5 non-intake:
df_key = windows_df[windows_df["key"].astype(str) == key].copy()

pos_rows = df_key[df_key["label"] == 1].head(5)
neg_rows = df_key[df_key["label"] == 0].head(5)

print(f"Found {len(pos_rows)} intake and {len(neg_rows)} non-intake 
windows for key {key}")

if len(pos_rows) < 5 or len(neg_rows) < 5:
    raise RuntimeError("Not enough intake or non-intake windows (need 
at least 5 of each).")

# Collect images:
img_infos = []  # list of (PIL.Image, title)

# First 5 intake:
for _, r in pos_rows.iterrows():
    t_mid = (float(r["start_sec"]) + float(r["end_sec"])) / 2.0
    p = frame_path_for_time(key, t_mid)
    img = Image.open(p).convert("RGB")
    img_infos.append((img, f"Intake @ {t_mid:.2f}s"))

# Then 5 non-intake:
for _, r in neg_rows.iterrows():
    t_mid = (float(r["start_sec"]) + float(r["end_sec"])) / 2.0
    p = frame_path_for_time(key, t_mid)
    img = Image.open(p).convert("RGB")
    img_infos.append((img, f"Non-intake @ {t_mid:.2f}s"))

# Plot as 2 rows x 5 columns:
rows, cols = 2, 5
n_imgs = len(img_infos)
assert n_imgs == rows * cols, f"Expected 10 images, got {n_imgs}"

fig, axes = plt.subplots(rows, cols, figsize=(cols * 3, rows * 3))

for idx, (img, title) in enumerate(img_infos):
    r = idx // cols
    c = idx % cols
    ax = axes[r, c]
    ax.imshow(img)
```

```
    ax.set_title(title, fontsize=8)
    ax.axis("off")

plt.tight_layout()
plt.show()
```

```
Candidate keys with both classes: 37
Using key: 20210523_202300
Found 5 intake and 5 non-intake windows for key 20210523_202300
```



Baseline RGB 3D-CNN Modeling

```
# Ste up configuration:


ROOT       = Path("/content/drive/MyDrive/eatsense")
LABELS_DIR = ROOT / "labels_v1"
FRAMES_DIR = ROOT / "frames"
CKPT_DIR   = ROOT / "checkpoints_rgb"
LOG_DIR    = ROOT / "logs_rgb"

IMG_EXT   = ".png"
FRAME_PAD = 6

for d in [CKPT_DIR, LOG_DIR]:
    d.mkdir(parents=True, exist_ok=True)

CONFIG: Dict[str, Any] = {
    "seed": 1981,
    "num_workers": 2,
    "pin_memory": True,
    "persistent_workers": True,
    # training
    "epochs": 10,
```

```
    "batch_size": 8,
    "lr": 3e-4,
    "weight_decay": 1e-5,
    "pos_class_weight": 2.0,    # BCE pos_weight
    "amp": True,
    "amp_dtype": "bf16",        # "bf16" or "fp16"
    # RGB clip settings
    "clip_len": 16,             # frames per window
    "img_size": 112,            # 112x112 input to 3D CNN
}

def set_seed(seed: int = 1981):
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False

set_seed(CONFIG["seed"])

DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Device: {DEVICE.type} | Torch: {torch.__version__}")

def get_logger(name="bitepulse_rgb",
               level=logging.INFO,
               log_file: Optional[Path] = LOG_DIR / "run_rgb3d.log"):
    logger = logging.getLogger(name)
    if logger.handlers:
        return logger
    logger.setLevel(level)
    fmt = logging.Formatter("%(asctime)s | %(levelname)s | %
(message)s")
    sh = logging.StreamHandler(stream=sys.stdout)
    sh.setFormatter(fmt)
    logger.addHandler(sh)
    if log_file:
        fh = logging.FileHandler(log_file)
        fh.setFormatter(fmt)
        logger.addHandler(fh)
    return logger

LOGGER = get_logger()

def make_amp_ctx():
    if not CONFIG["amp"] or DEVICE.type != "cuda":
        return nullcontext
    if CONFIG.get("amp_dtype", "bf16") == "bf16":
        return lambda: torch.amp.autocast("cuda",
dtype=torch.bfloat16)
```

```python
    else:
        return lambda: torch.amp.autocast("cuda", dtype=torch.float16)

AMP_CTX = make_amp_ctx()

SCALER = (
    torch.amp.GradScaler("cuda")
    if (CONFIG["amp"] and CONFIG.get("amp_dtype") == "fp16" and
DEVICE.type == "cuda")
    else None
)
```

```
Device: cuda | Torch: 2.9.0+cu126
```

```python
# Alright now let's load window table and frame index:


MANIFEST_PATH   = LABELS_DIR / "manifest_with_split.parquet"
FRAMES_IDX_PATH = LABELS_DIR / "frames_idx.parquet"
WINDOWS_PATH    = ROOT / "logs" / "windows_idx.parquet"

assert FRAMES_IDX_PATH.exists(), FRAMES_IDX_PATH
assert WINDOWS_PATH.exists(), WINDOWS_PATH

frames_ix  = pd.read_parquet(FRAMES_IDX_PATH)
windows_df = pd.read_parquet(WINDOWS_PATH)

if "time_sec" not in frames_ix.columns and {"frame",
"fps"}.issubset(frames_ix.columns):
    frames_ix = frames_ix.copy()
    frames_ix["time_sec"] = frames_ix["frame"] / frames_ix["fps"]

assert "time_sec" in frames_ix.columns, "frames_idx needs 'time_sec'
column"

print("windows_df:", windows_df.shape)
print("frames_ix :", frames_ix.shape)

# class balance by split:
print("\nLabel distribution by split:")
print(
    windows_df.groupby("split")["label"]
    .value_counts(normalize=True)
    .unstack()
    .fillna(0.0)
)
```

```
windows_df: (98582, 5)
frames_ix : (742887, 5)

Label distribution by split:
```

```
label           0         1
split
test   0.993999  0.006001
train  0.993255  0.006745
val    0.997309  0.002691
```

```
# Checking that keys in windows appear in frames_ix:
win_keys = set(windows_df["key"].astype(str))
frame_keys = set(frames_ix["key"].astype(str))
missing_keys = sorted(win_keys - frame_keys)
print(f"Unique keys in windows: {len(win_keys)}")
print(f"Unique keys in frames : {len(frame_keys)}")
print(f"Windows keys missing in frames_ix: {len(missing_keys)}")
if missing_keys:
    print("Example missing keys:", missing_keys[:10])
```

```
Unique keys in windows: 135
Unique keys in frames : 135
Windows keys missing in frames_ix: 0
```

```
# Now let's setup metric helpers and evaluation (similar to TCN)

def bin_metrics_from_logits_np(logits: np.ndarray,
                               targets: np.ndarray,
                               thr: float = 0.5):
    probs = 1.0 / (1.0 + np.exp(-logits))
    preds = (probs >= thr).astype(int)
    y     = targets.astype(int)

    tp = int(((preds == 1) & (y == 1)).sum())
    fp = int(((preds == 1) & (y == 0)).sum())
    fn = int(((preds == 0) & (y == 1)).sum())
    tn = int(((preds == 0) & (y == 0)).sum())

    prec = tp / (tp + fp + 1e-9)
    rec  = tp / (tp + fn + 1e-9)
    acc  = (tp + tn) / (tp + tn + fp + fn + 1e-9)
    f1   = 2 * prec * rec / (prec + rec + 1e-9)

    return {"acc": acc, "prec": prec, "rec": rec, "f1": f1,
            "tp": tp, "fp": fp, "fn": fn, "tn": tn}


def best_f1_threshold_with_floor(
    logits: np.ndarray,
    targets: np.ndarray,
    prec_floor: float = 0.10,
    rec_floor: float  = 0.10,
):
    y_true = targets.astype(int)
    probs  = 1.0 / (1.0 + np.exp(-logits.astype(np.float64)))
```

```python
    P, R, T = precision_recall_curve(y_true, probs)

    best_f1, best_thr = 0.0, 0.5
    best_tuple = (0.0, 0.0, 0, 0, 0)

    for p, r, t in zip(P[:-1], R[:-1], T):
        if (p < prec_floor) or (r < rec_floor):
            continue
        f1 = 2 * p * r / (p + r + 1e-12)
        if f1 > best_f1:
            thr = float(t)
            preds = (probs >= thr).astype(int)
            tp = int(((preds==1) & (y_true==1)).sum())
            fp = int(((preds==1) & (y_true==0)).sum())
            fn = int(((preds==0) & (y_true==1)).sum())
            best_f1, best_thr = f1, thr
            best_tuple = (p, r, tp, fp, fn)

    return best_f1, best_thr, best_tuple


def evaluate_window_model(logits_np: np.ndarray,
                          targets_np: np.ndarray,
                          label: str,
                          prec_floor: float = 0.10,
                          rec_floor: float = 0.10):
    y_true = targets_np.astype(int)
    f1_star, thr_star, _ = best_f1_threshold_with_floor(
        logits_np, y_true,
        prec_floor=prec_floor,
        rec_floor=rec_floor,
    )
    thr = float(thr_star)
    p_prob = 1.0 / (1.0 + np.exp(-logits_np.astype(np.float64)))
    y_pred = (p_prob >= thr).astype(int)

    tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()
    spec = tn / (tn + fp + 1e-12)

    metrics = {
        "threshold(F1*)": thr,
        "accuracy": accuracy_score(y_true, y_pred),
        "precision": precision_score(y_true, y_pred, zero_division=0),
        "recall": recall_score(y_true, y_pred, zero_division=0),
        "f1": f1_score(y_true, y_pred, zero_division=0),
        "specificity": spec,
        "balanced_acc": balanced_accuracy_score(y_true, y_pred),
        "mcc": matthews_corrcoef(y_true, y_pred),
        "roc_auc_overall": roc_auc_score(y_true, p_prob),
        "pr_auc_overall": average_precision_score(y_true, p_prob),
```

```
    }

    print(f"\n=== {label}: metrics @ best F1 threshold ===")
    for k, v in metrics.items():
        if isinstance(v, float):
            print(f"{k:>20s}: {v:0.3f}")
        else:
            print(f"{k:>20s}: {v}")

    print("\nConfusion Matrix (rows=actual, cols=pred):")
    print(np.array([[tn, fp], [fn, tp]]))
    print("\nClassification report:\n",
          classification_report(y_true, y_pred, digits=3,
zero_division=0))

    fig = plt.figure(figsize=(12, 4))

    cm = np.array([[tn, fp], [fn, tp]], dtype=float)
    cm_norm = cm / cm.sum(axis=1, keepdims=True).clip(1)

    ax1 = plt.subplot(1, 3, 1)
    im = ax1.imshow(cm_norm, vmin=0, vmax=1)
    ax1.set_title(f"{label} — Confusion @ thr={thr:.2f}")
    ax1.set_xlabel("Predicted")
    ax1.set_ylabel("Actual")
    ax1.set_xticks([0, 1]); ax1.set_xticklabels(["Not intake",
"Intake"])
    ax1.set_yticks([0, 1]); ax1.set_yticklabels(["Not intake",
"Intake"])
    for i in range(2):
        for j in range(2):
            ax1.text(j, i, f"{int(cm[i,j])}\n({cm_norm[i,j]*100:.1f}
%)",
                     ha="center", va="center",
                     color="white" if cm_norm[i,j] > 0.5 else "black",
                     fontsize=9)
    plt.colorbar(im, ax=ax1, fraction=0.046, pad=0.04)

    fpr, tpr, _ = roc_curve(y_true, p_prob)
    ax2 = plt.subplot(1, 3, 2)
    ax2.plot(fpr, tpr, label=f"AUC =
{metrics['roc_auc_overall']:.3f}")
    ax2.plot([0, 1], [0, 1], "--", linewidth=1)
    ax2.set_title(f"{label} — ROC")
    ax2.set_xlabel("False Positive Rate")
    ax2.set_ylabel("True Positive Rate")
    ax2.grid(True, alpha=0.3)
    ax2.legend(loc="lower right")

    P, R, _ = precision_recall_curve(y_true, p_prob)
```

```
    ax3 = plt.subplot(1, 3, 3)
    ax3.plot(R, P, label=f"AP = {metrics['pr_auc_overall']:.3f}")
    ax3.set_title(f"{label} — Precision Recall")
    ax3.set_xlabel("Recall")
    ax3.set_ylabel("Precision")
    ax3.grid(True, alpha=0.3)
    ax3.legend(loc="lower left")

    plt.tight_layout()
    plt.show()

    return {"metrics": metrics, "thr": thr, "probs": p_prob, "y_true":
y_true}

# Fixed RGB dataset + DataLoaders

IMG_EXT   = ".png"
FRAME_PAD = 6
CLIP_FPS  = float(FPS_TARGET) if "FPS_TARGET" in globals() else 15.0

class RGBWindowDataset(Dataset):
    """
    Each item is an RGB clip for a labeled window.
    x: (T, 3, H, W) in [0,1], y: scalar float32 (0 or 1).

    For each key we:
      1) List all PNG frames actually on disk.
      2) Map window times [w0, w1) to frame indices via CLIP_FPS.
      3) Clamp indices into [0, n_frames-1] so we never hit a missing
file.
    """
    def __init__(self,
                 windows: pd.DataFrame,
                 clip_len: int = 16,
                 img_size: int = 112,
                 fps: float = CLIP_FPS):
        self.df       = windows.reset_index(drop=True)
        self.clip_len = int(clip_len)
        self.img_size = int(img_size)
        self.fps      = float(fps)

    def _load_clip(self, key: str, w0: float, w1: float):
        key = str(key)
        kdir = FRAMES_DIR / key
        files = sorted(kdir.glob(f"*{IMG_EXT}"))

        if not files:
            raise RuntimeError(f"No frame files found for key={key} in
{kdir}")
```

```
        n_frames = len(files)

        #
        times = np.linspace(w0, w1, num=self.clip_len, endpoint=False)

        frames = []
        for t in times:

            idx = int(round(t * self.fps))
            idx = max(0, min(n_frames - 1, idx))
            path = files[idx]

            img = Image.open(path).convert("RGB")
            img = img.resize((self.img_size, self.img_size))
            x = np.asarray(img, dtype=np.float32) / 255.0
            x = np.transpose(x, (2, 0, 1))
            frames.append(x)

        arr = np.stack(frames, axis=0)
        return arr.astype(np.float32)

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

    def __getitem__(self, i: int):
        r = self.df.iloc[i]
        key   = r["key"]
        w0    = float(r["start_sec"])
        w1    = float(r["end_sec"])
        label = float(r["label"])

        clip = self._load_clip(key, w0, w1)
        x    = torch.from_numpy(clip)
        y    = torch.tensor(label, dtype=torch.float32)
        return x, y


def make_rgb_loader(windows_df, split: str, balance: bool = True):
    # take split:
    df_split = windows_df[windows_df["split"] ==
split].reset_index(drop=True)

    # PNGs location:
    keys_with_frames = []
    for p in FRAMES_DIR.iterdir():
        if p.is_dir() and any(p.glob(f"*{IMG_EXT}")):
            keys_with_frames.append(p.name)
    keys_with_frames = set(keys_with_frames)

    # filter windows:
```

```
    mask = df_split["key"].astype(str).isin(keys_with_frames)
    n_before = len(df_split)
    df_split = df_split[mask].reset_index(drop=True)
    n_after = len(df_split)
    print(f"[{split}] windows with frames: {n_after}/{n_before} "
          f"(dropped {n_before - n_after})")

    # dataset:
    ds = RGBWindowDataset(
        df_split,
        clip_len=CONFIG["clip_len"],
        img_size=CONFIG["img_size"],
        fps=CLIP_FPS,
    )

    # class balancing:
    if balance and split == "train":
        labels = ds.df["label"].astype(int).values
        counts = np.bincount(labels, minlength=2).astype(np.float32)
        weights = 1.0 / (counts[labels] + 1e-9)
        sampler = WeightedRandomSampler(
            weights.tolist(),
            num_samples=len(weights),
            replacement=True,
        )
        shuffle = False
    else:
        sampler = None
        shuffle = (split == "train")

    loader = DataLoader(
        ds,
        batch_size=CONFIG["batch_size"],
        shuffle=shuffle if sampler is None else False,
        sampler=sampler,
        num_workers=CONFIG["num_workers"],
        pin_memory=CONFIG["pin_memory"],
        persistent_workers=CONFIG["persistent_workers"] if
CONFIG["num_workers"] > 0 else False,
        drop_last=False,
    )
    return loader, ds


# Loaders:
train_rgb_loader, train_rgb_ds = make_rgb_loader(windows_df, "train",
balance=True)
val_rgb_loader,   val_rgb_ds   = make_rgb_loader(windows_df, "val",
balance=False)
```

```
# Sanity check:
clip, label = train_rgb_ds[0]
print("Single clip shape:", clip.shape, "label:", label)
print("clip stats: min", float(clip.min()),
      "max", float(clip.max()),
      "mean", float(clip.mean()),
      "std", float(clip.std()))

batch_x, batch_y = next(iter(train_rgb_loader))
print("Batch shape:", batch_x.shape)   # expect (B,T,3,112,112)
print("Batch stats:",
      float(batch_x.min()),
      float(batch_x.max()),
      float(batch_x.mean()),
      float(batch_x.std()))
```

```
[train] windows with frames: 7268/62568 (dropped 55300)
[val] windows with frames: 2565/21184 (dropped 18619)
Single clip shape: torch.Size([16, 3, 112, 112]) label: tensor(0.)
clip stats: min 0.0 max 1.0 mean 0.45068100094795227 std
0.24670635163784027
Batch shape: torch.Size([8, 16, 3, 112, 112])
Batch stats: 0.0 1.0 0.47496291995048523 0.25107431411743164
```

Baseline PyTorch 3D CNN - Training

```
class RGB3DNet(nn.Module):
    """
    3D CNN for RGB clips.
    Input:  x (B, T, 3, H, W)
    Output: logits of shape (B,) for intake vs non-intake.
    """
    def __init__(self):
        super().__init__()
        # we'll permute to (B, 3, T, H, W) in forward
        self.features = nn.Sequential(
            nn.Conv3d(3, 32, kernel_size=(3, 7, 7),
                      stride=(1, 2, 2), padding=(1, 3, 3),
bias=False),
            nn.BatchNorm3d(32),
            nn.ReLU(inplace=True),
            nn.MaxPool3d(kernel_size=(1, 3, 3),
                         stride=(1, 2, 2), padding=(0, 1, 1)),

            nn.Conv3d(32, 64, kernel_size=3,
                      stride=1, padding=1, bias=False),
            nn.BatchNorm3d(64),
            nn.ReLU(inplace=True),
            nn.MaxPool3d(kernel_size=(2, 2, 2),
```

```
                        stride=(2, 2, 2)),

            nn.Conv3d(64, 128, kernel_size=3,
                      stride=1, padding=1, bias=False),
            nn.BatchNorm3d(128),
            nn.ReLU(inplace=True),
            nn.MaxPool3d(kernel_size=(2, 2, 2),
                         stride=(2, 2, 2)),
        )

        self.pool = nn.AdaptiveAvgPool3d(1)   # (B, C, 1, 1, 1)
        self.fc   = nn.Linear(128, 1)

    def forward(self, x):
        # x: (B, T, 3, H, W) -> (B, 3, T, H, W)
        x = x.permute(0, 2, 1, 3, 4)
        x = self.features(x)
        x = self.pool(x).view(x.size(0), -1)
        x = self.fc(x)
        return x.squeeze(-1)   # (B,)


def run_rgb_epoch(model, loader, optimizer=None):
    """
    One pass over a DataLoader.
    If optimizer is None -> eval mode, no grads.
    Returns: avg_loss, metrics_dict, logits_np, targets_np
    """
    train_mode = optimizer is not None
    model.train(train_mode)

    pos_w = torch.tensor(
        [CONFIG.get("pos_class_weight", 1.0)],
        device=DEVICE, dtype=torch.float32,
    )
    criterion = nn.BCEWithLogitsLoss(pos_weight=pos_w)

    total_loss, n_seen = 0.0, 0
    all_logits, all_targets = [], []

    for x, y in tqdm(loader, desc="batches", leave=False):
        # x: (B,T,3,H,W), y: scalar {0,1}
        x = x.to(DEVICE, non_blocking=True).float()
        y = y.to(DEVICE, non_blocking=True).float()  # (B,)

        if train_mode:
            optimizer.zero_grad(set_to_none=True)

        with AMP_CTX():
            logits = model(x)            # (B,)
```

```
            loss   = criterion(logits, y)

        if train_mode:
            loss.backward()
            optimizer.step()

        bs = x.size(0)
        total_loss += float(loss.item()) * bs
        n_seen     += bs

        all_logits.append(logits.detach().cpu())
        all_targets.append(y.detach().cpu())

    # ---- FIX: cast bfloat16 -> float32 before converting to NumPy
----
    if len(all_logits) == 0:
        return 0.0, {"acc": 0, "prec": 0, "rec": 0, "f1": 0},
np.array([]), np.array([])

    logits_cat  = torch.cat(all_logits, dim=0).to(torch.float32)
    targets_cat = torch.cat(all_targets, dim=0).to(torch.float32)

    logits_np  = logits_cat.numpy()
    targets_np = targets_cat.numpy().astype(int)

    metrics = bin_metrics_from_logits_np(logits_np, targets_np,
thr=0.5)

    return total_loss / max(1, n_seen), metrics, logits_np, targets_np



def train_rgb3d(train_loader, val_loader):
    model = RGB3DNet().to(DEVICE)

    optimizer = torch.optim.AdamW(
        model.parameters(),
        lr=CONFIG["lr"],
        weight_decay=CONFIG["weight_decay"],
    )

    scheduler = torch.optim.lr_scheduler.OneCycleLR(
        optimizer,
        max_lr=8e-4,
        steps_per_epoch=len(train_loader),
        epochs=CONFIG["epochs"],
    )

    best_val_f1 = -1.0
    best_state  = None
```

```python
    for epoch in range(CONFIG["epochs"]):
        LOGGER.info(f"[RGB3D] epoch {epoch+1}/{CONFIG['epochs']}")

        tr_loss, tr_m, _, _ = run_rgb_epoch(model, train_loader,
                                            optimizer=optimizer)
        try:
            scheduler.step()
        except Exception:
            pass

        va_loss, va_m, va_logits_np, va_y_np = run_rgb_epoch(
            model, val_loader, optimizer=None
        )

        f1_star, thr_star, _ = best_f1_threshold_with_floor(
            va_logits_np, va_y_np
        )

        LOGGER.info(
            f"train loss {tr_loss:.4f} | "
            f"acc {tr_m['acc']:.3f} prec {tr_m['prec']:.3f} "
            f"rec {tr_m['rec']:.3f} f1 {tr_m['f1']:.3f}"
        )
        LOGGER.info(
            f" val  loss {va_loss:.4f} | best_f1 {f1_star:.3f} "
            f"thr {thr_star:.2f}"
        )

        if f1_star > best_val_f1:
            best_val_f1 = f1_star
            best_state = {
                "model": model.state_dict(),
                "val_f1": float(best_val_f1),
                "best_thr": float(thr_star),
            }
            torch.save(best_state, CKPT_DIR / "rgb3d_best.pt")
            LOGGER.info(
                f"✓ saved new best (f1={best_val_f1:.3f} @
thr={thr_star:.2f})"
            )

    LOGGER.info(f"[RGB3D] best val f1={best_val_f1:.3f}")
    return model


model_rgb3d = train_rgb3d(train_rgb_loader, val_rgb_loader)
```

```
2025-11-27 20:34:21,125 | INFO | [RGB3D] epoch 1/10
```

```
INFO:bitepulse_rgb:[RGB3D] epoch 1/10
```

```
{"model_id":"b770a33faffb413791b15d7dd8b706ec","version_major":2,"version_minor":0}

{"model_id":"551ced337ee44715bd6cccd703ce8ee8","version_major":2,"version_minor":0}

2025-11-27 21:54:39,647 | INFO | train loss 0.4772 | acc 0.816 prec 0.751 rec 0.952 f1 0.839

INFO:bitepulse_rgb:train loss 0.4772 | acc 0.816 prec 0.751 rec 0.952 f1 0.839

2025-11-27 21:54:39,649 | INFO |  val  loss 0.7800 | best_f1 0.134 thr 0.73

INFO:bitepulse_rgb: val  loss 0.7800 | best_f1 0.134 thr 0.73

2025-11-27 21:54:40,498 | INFO | ✓ saved new best (f1=0.134 @ thr=0.73)

INFO:bitepulse_rgb:✓ saved new best (f1=0.134 @ thr=0.73)

2025-11-27 21:54:40,500 | INFO | [RGB3D] epoch 2/10

INFO:bitepulse_rgb:[RGB3D] epoch 2/10

{"model_id":"1850aec37d70459cb02d6381398ba5a1","version_major":2,"version_minor":0}

{"model_id":"69c8bf727534459b836760ade176c240","version_major":2,"version_minor":0}

2025-11-27 22:33:00,060 | INFO | train loss 0.3072 | acc 0.903 prec 0.855 rec 0.973 f1 0.910

INFO:bitepulse_rgb:train loss 0.3072 | acc 0.903 prec 0.855 rec 0.973 f1 0.910

2025-11-27 22:33:00,062 | INFO |  val  loss 0.3229 | best_f1 0.188 thr 0.56

INFO:bitepulse_rgb: val  loss 0.3229 | best_f1 0.188 thr 0.56

2025-11-27 22:33:00,073 | INFO | ✓ saved new best (f1=0.188 @ thr=0.56)

INFO:bitepulse_rgb:✓ saved new best (f1=0.188 @ thr=0.56)

2025-11-27 22:33:00,074 | INFO | [RGB3D] epoch 3/10

INFO:bitepulse_rgb:[RGB3D] epoch 3/10

{"model_id":"4bdb3731fc674b4a9a022705f8f0c4f9","version_major":2,"version_minor":0}
```

{"model_id":"b4caf189f15840168d3fe0d366d6edcc","version_major":2,"version_minor":0}

```
2025-11-27 23:09:23,452 | INFO | train loss 0.2176 | acc 0.941 prec 0.902 rec 0.991 f1 0.944
INFO:bitepulse_rgb:train loss 0.2176 | acc 0.941 prec 0.902 rec 0.991 f1 0.944
2025-11-27 23:09:23,454 | INFO |  val  loss 0.1337 | best_f1 0.186 thr 0.03
INFO:bitepulse_rgb: val  loss 0.1337 | best_f1 0.186 thr 0.03
2025-11-27 23:09:23,455 | INFO | [RGB3D] epoch 4/10
INFO:bitepulse_rgb:[RGB3D] epoch 4/10
```

{"model_id":"41d49a772acf45ddae254868e34b8512","version_major":2,"version_minor":0}

{"model_id":"82fc9bec6a17417fadd456a055ef5e18","version_major":2,"version_minor":0}

```
2025-11-27 23:45:05,509 | INFO | train loss 0.1597 | acc 0.966 prec 0.943 rec 0.992 f1 0.967
INFO:bitepulse_rgb:train loss 0.1597 | acc 0.966 prec 0.943 rec 0.992 f1 0.967
2025-11-27 23:45:05,510 | INFO |  val  loss 3.3173 | best_f1 0.163 thr 1.00
INFO:bitepulse_rgb: val  loss 3.3173 | best_f1 0.163 thr 1.00
2025-11-27 23:45:05,512 | INFO | [RGB3D] epoch 5/10
INFO:bitepulse_rgb:[RGB3D] epoch 5/10
```

{"model_id":"a751535ffea44d29ab45de000e861568","version_major":2,"version_minor":0}

{"model_id":"346034f0692244adbee9cf88a51d34ee","version_major":2,"version_minor":0}

```
2025-11-28 00:21:16,923 | INFO | train loss 0.1284 | acc 0.978 prec 0.960 rec 0.997 f1 0.978
INFO:bitepulse_rgb:train loss 0.1284 | acc 0.978 prec 0.960 rec 0.997 f1 0.978
2025-11-28 00:21:16,924 | INFO |  val  loss 0.2908 | best_f1 0.137 thr 0.65
```

```
INFO:bitepulse_rgb: val  loss 0.2908 | best_f1 0.137 thr 0.65

2025-11-28 00:21:16,926 | INFO | [RGB3D] epoch 6/10

INFO:bitepulse_rgb:[RGB3D] epoch 6/10
```
{"model_id":"7d106fb784334f35b78be771648dab9b","version_major":2,"version_minor":0}

{"model_id":"6fdc11a309974c28bcc58e83352475ca","version_major":2,"version_minor":0}
```
2025-11-28 00:56:34,979 | INFO | train loss 0.1022 | acc 0.984 prec 0.969 rec 0.999 f1 0.984

INFO:bitepulse_rgb:train loss 0.1022 | acc 0.984 prec 0.969 rec 0.999 f1 0.984

2025-11-28 00:56:34,980 | INFO |  val  loss 0.1300 | best_f1 0.153 thr 0.15

INFO:bitepulse_rgb: val  loss 0.1300 | best_f1 0.153 thr 0.15

2025-11-28 00:56:34,981 | INFO | [RGB3D] epoch 7/10

INFO:bitepulse_rgb:[RGB3D] epoch 7/10
```
{"model_id":"0d9b5bd3d4e244d8945be0c39412bf41","version_major":2,"version_minor":0}

{"model_id":"c3c6f3f3dcb545649d78fbdf7b56f29a","version_major":2,"version_minor":0}
```
2025-11-28 01:31:01,353 | INFO | train loss 0.0903 | acc 0.984 prec 0.971 rec 0.998 f1 0.984

INFO:bitepulse_rgb:train loss 0.0903 | acc 0.984 prec 0.971 rec 0.998 f1 0.984

2025-11-28 01:31:01,354 | INFO |  val  loss 0.1369 | best_f1 0.161 thr 0.17

INFO:bitepulse_rgb: val  loss 0.1369 | best_f1 0.161 thr 0.17

2025-11-28 01:31:01,355 | INFO | [RGB3D] epoch 8/10

INFO:bitepulse_rgb:[RGB3D] epoch 8/10
```
{"model_id":"dd9a9fedb9f8414d9cdeca71407f6d2a","version_major":2,"version_minor":0}

{"model_id":"21537472efb54629aa0f5c6b61dc938b","version_major":2,"version_minor":0}

```
2025-11-28 02:06:02,741 | INFO | train loss 0.0801 | acc 0.984 prec 0.970 rec 0.998 f1 0.984

INFO:bitepulse_rgb:train loss 0.0801 | acc 0.984 prec 0.970 rec 0.998 f1 0.984

2025-11-28 02:06:02,742 | INFO |  val  loss 1.4249 | best_f1 0.000 thr 0.50

INFO:bitepulse_rgb: val  loss 1.4249 | best_f1 0.000 thr 0.50

2025-11-28 02:06:02,743 | INFO | [RGB3D] epoch 9/10

INFO:bitepulse_rgb:[RGB3D] epoch 9/10
```

{"model_id":"376845b9df28452f90026598589ca38b","version_major":2,"version_minor":0}

{"model_id":"f49d736915754191890553eec1bfd63c","version_major":2,"version_minor":0}

```
2025-11-28 02:41:38,777 | INFO | train loss 0.0639 | acc 0.989 prec 0.978 rec 1.000 f1 0.989

INFO:bitepulse_rgb:train loss 0.0639 | acc 0.989 prec 0.978 rec 1.000 f1 0.989

2025-11-28 02:41:38,778 | INFO |  val  loss 0.1281 | best_f1 0.175 thr 0.17

INFO:bitepulse_rgb: val  loss 0.1281 | best_f1 0.175 thr 0.17

2025-11-28 02:41:38,779 | INFO | [RGB3D] epoch 10/10

INFO:bitepulse_rgb:[RGB3D] epoch 10/10
```

{"model_id":"333396442d4b472aadae0e9a1bff368c","version_major":2,"version_minor":0}

{"model_id":"287c258b8d8c44ddb036e0db49d3e587","version_major":2,"version_minor":0}

```
2025-11-28 03:18:21,930 | INFO | train loss 0.0604 | acc 0.989 prec 0.979 rec 1.000 f1 0.989

INFO:bitepulse_rgb:train loss 0.0604 | acc 0.989 prec 0.979 rec 1.000 f1 0.989

2025-11-28 03:18:21,931 | INFO |  val  loss 0.1285 | best_f1 0.174 thr 0.12

INFO:bitepulse_rgb: val  loss 0.1285 | best_f1 0.174 thr 0.12

2025-11-28 03:18:21,933 | INFO | [RGB3D] best val f1=0.188
```

```
INFO:bitepulse_rgb:[RGB3D] best val f1=0.188
```

Baseline PyTorch 3D CNN - Performance Evaluation

```
best_ckpt_path = CKPT_DIR / "rgb3d_best.pt"
print("Checkpoint:", best_ckpt_path, "| exists:",
best_ckpt_path.exists())

ckpt = torch.load(best_ckpt_path, map_location=DEVICE)

model_rgb3d_best = RGB3DNet().to(DEVICE)
model_rgb3d_best.load_state_dict(ckpt["model"])
model_rgb3d_best.eval()

print(f"Best val F1 (saved in ckpt): {ckpt.get('val_f1', None)}  @
thr={ckpt.get('best_thr', None)}")
```

```
Checkpoint:
/content/drive/MyDrive/eatsense/checkpoints_rgb/rgb3d_best.pt |
exists: True
Best val F1 (saved in ckpt): 0.18803418803373279  @
thr=0.5640984800226174
```

```
# Best checkpoint model:
val_loss_rgb, val_m_rgb, val_logits_rgb, val_y_rgb = run_rgb_epoch(
    model_rgb3d_best,
    val_rgb_loader,
    optimizer=None,
)

print("=== RGB3D baseline: quick val metrics @ thr=0.5 ===")
for k, v in val_m_rgb.items():
    print(f"{k:>10s}: {v:.3f}")
print("val_loss:", val_loss_rgb)
```

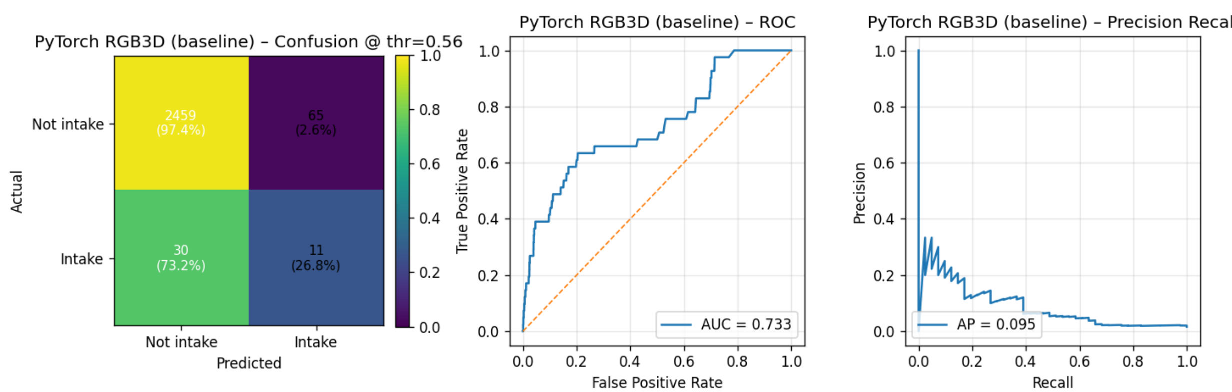
{"model_id":"ed8b64fa7e6b4eadb4549c1c651cb5c4","version_major":2,"vers
ion_minor":0}

```
=== RGB3D baseline: quick val metrics @ thr=0.5 ===
       acc: 0.948
      prec: 0.122
       rec: 0.366
        f1: 0.183
        tp: 15.000
        fp: 108.000
        fn: 26.000
        tn: 2416.000
val_loss: 0.32293026334593405
```

```
rgb3d_eval = evaluate_window_model(
    logits_np  = val_logits_rgb,
```

```
    targets_np = val_y_rgb,
    label      = "PyTorch RGB3D (baseline)",
    prec_floor = 0.10,
    rec_floor  = 0.10,
)
```

```
=== PyTorch RGB3D (baseline): metrics @ best F1 threshold ===
      threshold(F1*): 0.564
            accuracy: 0.963
           precision: 0.145
              recall: 0.268
                  f1: 0.188
         specificity: 0.974
        balanced_acc: 0.621
                 mcc: 0.179
     roc_auc_overall: 0.733
      pr_auc_overall: 0.095

Confusion Matrix (rows=actual, cols=pred):
[[2459   65]
 [  30   11]]

Classification report:
              precision    recall  f1-score   support

           0      0.988     0.974     0.981      2524
           1      0.145     0.268     0.188        41

    accuracy                          0.963      2565
   macro avg      0.566     0.621     0.585      2565
weighted avg      0.974     0.963     0.968      2565
```



3D-CNN - hyperband Search

```
# TF GPU config:
import tensorflow as tf
```

```python
gpus = tf.config.list_physical_devices("GPU")
for g in gpus:
    try:
        tf.config.experimental.set_memory_growth(g, True)
    except Exception:
        pass
print("TF GPUs:", gpus)
```

```
TF GPUs: [PhysicalDevice(name='/physical_device:GPU:0',
device_type='GPU')]
```

```python
import math

class RGBWindowSequence(tf.keras.utils.Sequence):
    """
    Keras data generator: yields (X, y) batches for window-level RGB
clips.

    X shape: (B, T, H, W, 3) in [0,1]
    y shape: (B, 1) float32 {0,1}
    """
    def __init__(self,
                 windows: pd.DataFrame,
                 batch_size: int = 4,
                 clip_len: int = 16,
                 img_size: int = 112,
                 fps: float = CLIP_FPS):
        self.df         = windows.reset_index(drop=True)
        self.batch_size = int(batch_size)
        self.clip_len   = int(clip_len)
        self.img_size   = int(img_size)
        self.fps        = float(fps)

    def __len__(self):
        return math.ceil(len(self.df) / self.batch_size)

    def _load_clip(self, key: str, w0: float, w1: float):
        key = str(key)
        kdir = FRAMES_DIR / key
        files = sorted(kdir.glob(f"*{IMG_EXT}"))

        if not files:
            raise RuntimeError(f"No frame files found for key={key} in
{kdir}")

        n_frames = len(files)
        times = np.linspace(w0, w1, num=self.clip_len, endpoint=False)

        frames = []
```

```
        for t in times:
            idx = int(round(t * self.fps))
            idx = max(0, min(n_frames - 1, idx))
            path = files[idx]

            img = Image.open(path).convert("RGB")
            img = img.resize((self.img_size, self.img_size))
            x = np.asarray(img, dtype=np.float32) / 255.0  # H,W,3
            frames.append(x)

        arr = np.stack(frames, axis=0)              # T,H,W,3
        return arr.astype(np.float32)

    def __getitem__(self, idx: int):
        b_start = idx * self.batch_size
        b_end   = min(len(self.df), (idx + 1) * self.batch_size)
        batch_df = self.df.iloc[b_start:b_end]

        B = len(batch_df)
        X = np.zeros((B, self.clip_len, self.img_size, self.img_size,
3),
                     dtype=np.float32)
        y = np.zeros((B, 1), dtype=np.float32)

        for i, (_, r) in enumerate(batch_df.iterrows()):
            clip = self._load_clip(
                key   = r["key"],
                w0    = float(r["start_sec"]),
                w1    = float(r["end_sec"]),
            )  # (T,H,W,3)
            X[i] = clip
            y[i, 0] = float(r["label"])

        return X, y

keys_with_frames = []
for p in FRAMES_DIR.iterdir():
    if p.is_dir() and any(p.glob(f"*{IMG_EXT}")):
        keys_with_frames.append(p.name)
keys_with_frames = set(keys_with_frames)
print("Keys with frames:", len(keys_with_frames))

# Filter windows for train / val:
train_df_keras = windows_df[
    (windows_df["split"] == "train") &
    (windows_df["key"].astype(str).isin(keys_with_frames))
].reset_index(drop=True)

val_df_keras = windows_df[
    (windows_df["split"] == "val") &
```

```python
    (windows_df["key"].astype(str).isin(keys_with_frames))
].reset_index(drop=True)

print("Keras train windows:", train_df_keras.shape)
print("Keras val   windows:", val_df_keras.shape)

KERAS_BATCH = 4

train_seq = RGBWindowSequence(
    train_df_keras,
    batch_size=KERAS_BATCH,
    clip_len=CONFIG["clip_len"],
    img_size=CONFIG["img_size"],
    fps=CLIP_FPS,
)
val_seq = RGBWindowSequence(
    val_df_keras,
    batch_size=KERAS_BATCH,
    clip_len=CONFIG["clip_len"],
    img_size=CONFIG["img_size"],
    fps=CLIP_FPS,
)

# Class weights to handle imbalance:
train_labels_np = train_df_keras["label"].astype(int).values
counts = np.bincount(train_labels_np, minlength=2).astype(float)
total = counts.sum()
class_weight = {
    0: float(total / (2.0 * counts[0])),
    1: float(total / (2.0 * counts[1] + 1e-9)),
}
print("Class counts:", counts, "class_weight:", class_weight)
```

```
Keys with frames: 17
Keras train windows: (7268, 5)
Keras val   windows: (2565, 5)
Class counts: [7186.   82.] class_weight: {0: 0.505705538547175, 1:
44.317073170461484}
```

```python
Xb, yb = train_seq[0]
print("Batch X shape:", Xb.shape)  # (B,T,H,W,3)
print("Batch y shape:", yb.shape, "labels:", yb[:8].ravel())
print("Batch stats: min", Xb.min(), "max", Xb.max(), "mean",
Xb.mean(), "std", Xb.std())
```

```
Batch X shape: (4, 16, 112, 112, 3)
Batch y shape: (4, 1) labels: [0. 0. 0. 0.]
Batch stats: min 0.0 max 1.0 mean 0.45622057 std 0.24634425
```

```python
def build_rgb3d_keras(hp):
    clip_len = CONFIG["clip_len"]
```

```
    img_size = CONFIG["img_size"]

    inputs = tf.keras.Input(
        shape=(clip_len, img_size, img_size, 3),
        name="rgb_clip",
    )

    # Hyperparameters
    base_filters = hp.Choice("base_filters", [32, 48, 64])
    num_blocks   = hp.Int("num_blocks", min_value=2, max_value=4,
step=1)
    dropout_rate = hp.Float("dropout", 0.0, 0.5, step=0.1)
    lr           = hp.Float("lr", 1e-4, 5e-3, sampling="log")

    x = inputs
    filters = base_filters
    for b in range(num_blocks):
        x = tf.keras.layers.Conv3D(
            filters,
            kernel_size=(3, 3, 3),
            strides=(1, 2, 2) if b == 0 else (1, 1, 1),
            padding="same",
            use_bias=False,
        )(x)
        x = tf.keras.layers.BatchNormalization()(x)
        x = tf.keras.layers.Activation("relu")(x)
        x = tf.keras.layers.MaxPool3D(pool_size=(1, 2, 2))(x)
        filters *= 2  # double channels each block

    x = tf.keras.layers.GlobalAveragePooling3D()(x)
    if dropout_rate > 0.0:
        x = tf.keras.layers.Dropout(dropout_rate)(x)

    outputs = tf.keras.layers.Dense(1, activation="sigmoid")(x)

    model = tf.keras.Model(inputs, outputs, name="rgb3d_keras_hb")

    model.compile(
        optimizer=tf.keras.optimizers.Adam(learning_rate=lr),
        loss=tf.keras.losses.BinaryCrossentropy(from_logits=False),
        metrics=[
            tf.keras.metrics.BinaryAccuracy(name="acc"),
            tf.keras.metrics.Precision(name="prec"),
            tf.keras.metrics.Recall(name="rec"),
            tf.keras.metrics.AUC(name="pr_auc", curve="PR"),
        ],
    )
    return model
```

```
import keras_tuner as kt

tuner_dir = ROOT / "keras_tuner_rgb"
tuner_dir.mkdir(parents=True, exist_ok=True)

tuner = kt.Hyperband(
    build_rgb3d_keras,
    objective=kt.Objective("val_pr_auc", direction="max"),
    max_epochs=6,
    factor=2,
    directory=str(tuner_dir),
    project_name="rgb3d_hyperband",
    overwrite=True,
)

early_stop = tf.keras.callbacks.EarlyStopping(
    monitor="val_pr_auc",
    mode="max",
    patience=2,
    restore_best_weights=True,
)

tuner.search(
    train_seq,
    validation_data=val_seq,
    epochs=6,
    callbacks=[early_stop],
    class_weight=class_weight,
)
```

```
Trial 1 Complete [04h 35m 19s]
val_pr_auc: 0.02312251180410385

Best val_pr_auc So Far: 0.02312251180410385
Total elapsed time: 04h 35m 19s

Search: Running Trial #2

Value             |Best Value So Far |Hyperparameter
48                |64                |base_filters
2                 |2                 |num_blocks
0.2               |0                 |dropout
0.00027996        |0.0039115         |lr
2                 |2                 |tuner/epochs
0                 |0                 |tuner/initial_epoch
2                 |2                 |tuner/bracket
0                 |0                 |tuner/round

Epoch 1/2
```

```
  31/1817 ━━━━━━━━━━━━━━━━━━━━ 1:56:40 4s/step - acc: 1.0000 - loss: 
0.0703 - pr_auc: 0.0000e+00 - prec: 0.0000e+00 - rec: 0.0000e+00
```

3D-CNN - hyperband Result

Given the time and compute limits for the capstone, I stopped Hyperband after the first full trial as shown above and pivoted to the more efficient MS-TCN architecture with frame-level supervision instead of continuing the full RGB 3D-CNN search.