Leukemia Project – Segmentation ➜ Feature Extraction ➜ Prediction

This repo contains a complete, reproducible pipeline to:

1. **Segment** microscopy images with Cellpose (plus optional fallback),
2. **Extract features** (skimage *and* Pyfeats) from segmented masks using channel-aware column names, and
3. **Predict** with an existing XGBoost checkpoint while transparently remapping legacy feature names to the new convention.

Quick Start

```
# 1) Segment (from a TXT resources file)
python segment_cells_refactored.py /path/to/resources.txt

# 2) Extract features (from a TXT resources file)
python Image_Features_Extraction_cif.py /path/to/resources_features.txt
python Pyfeats_Image_Features_Extraction_cif.py /path/to/
resources_features.txt

# 3) Predict (XGBoost)
python prediction.py
  --features "/path/to/skimage_<EXPERIMENT>_<SUB>.csv"
  --model   "/path/to/xgb_checkpoint.pkl"
  --feature-list "/path/to/feature_names.txt"   # only when model lacks names
```

Requirements

- Python 3.9+
- Packages: `numpy`, `pandas`, `scikit-image`, `imageio`, `matplotlib`, `xgboost`, `joblib`
- **Segmentation**: `cellpose` (GPU optional)
- **Pyfeats features**: an importable backend function (see below) and `concurrent.futures` (standard library)

  Tip: create a virtual environment and `pip install -r requirements.txt` (you can export one from your working env).

Data Layout (expected)

The segmentation script scans **experiment** folders and their subfolders for `.tif/.tiff` images.

```
<path_for_raw_images>/
  ├─ Human normal donors/              # example experiment name
  │   ├─ SetA/                         # arbitrary subfolder(s)
  │   │   ├─ 11_Ch1_....tif
  │   │   ├─ 11_Ch5_....tif
  │   │   └─ 11_Ch6_....tif
  │   └─ ...
  └─ WT control experiment/
      ├─ GFP_pos/
      └─ GFP_neg/
```

After segmentation, standardized output names will contain channel **roles**: `BF`, `Nucleus`, `DF` (and any additional roles you configure).

Files that differ **only by channel tag** (e.g., `11_Ch1*`, `11_Ch5*`, `11_Ch6*` or `11_BF*`, `11_Nucleus*`, `11_DF*`) are considered one **image set**.

---

1) Segmentation

Script: `segment_cells_refactored.py`

What it does

- Detects channels by **tags**, then **standardizes output filenames** by replacing the tag with the role name:
- `*_Ch1*.tif` → `*_BF*.tif`
- `*_Ch5*.tif` → `*_Nucleus*.tif`
- `*_Ch6*.tif` → `*_DF*.tif`
- If **BF & DF** both exist for a set, DF **reuses** the BF mask when available.
- Supports **additional roles** (e.g., `Actin`) processed like BF.
- **Per-set completeness rule:** if any *required* core role is missing for a set, the entire set is **skipped** and logged (no partial processing).
- *Required core roles* are inferred from which of `bf_tags`, `nucleus_tags`, `df_tags` appear in your resources file.
- **Idempotent re-runs:** it won't reprocess sets whose expected outputs already exist.
- **Dry run:** reports planned work and skipped sets; **no images** are written.
- **Fallback:** optional call into your legacy `Segmenting_Image.Segmentation_Program` if Cellpose fails.

Resources (TXT) — example

Create a `resources.txt`:

```
# Required
path_for_raw_images=/abs/path/to/Raw_images
experiments_types=Human normal donors,WT control experiment
save_path_segmented_images=/abs/path/to/Segmented_images_CellPose
```

```
# Channel tags (used to detect files & standardize names)
bf_tags=BF,Ch1
df_tags=DF,Ch6
nucleus_tags=Nucleus,Ch5

# Optional extra roles (each needs <role>_tags)
# additional_roles=Actin
# Actin_tags=Actin

# Behavior
use_optional_fallback=true
# dry-run only reports planned outputs & skipped sets
# dry_run=true
```

Run

```
python segment_cells_refactored.py /path/to/resources.txt
```

Outputs

Under `save_path_segmented_images/`:

```
Tiff_images/<rel_dir>/*.tif      # masked image outputs (standardized names)
Png_images/<rel_dir>/*.png       # quick overlay previews

fallback_segmentation_warning_<k>.txt  # images where fallback was used
(success)
segmentation_failed_<k>.txt            # images where Cellpose+fallback
failed
successful_segmentations_<k>.txt       # every successfully saved output
channels_unavailability_<k>.txt        # per-set: missing required core
channels (skipped)
planned_outputs_<k>.txt                # non-dry planned work
already_processed_sets_<k>.txt         # sets entirely complete on entry
area_flagged_corrupted_segmentations_paths_New_<k>.txt  # relaxed area
threshold notes
manifest_<k>.csv                       # per-image action log (processed/
reused/skip)

dry_run_report_<k>.txt                 # only when dry_run=true
```

The run index `<k>` auto-increments (`_0`, `_1`, ...). Re-runs never overwrite prior logs.

---

2) Feature Extraction

Two scripts produce **per (EXPERIMENT, SUBFOLDER)** CSVs with role-suffixed feature names. Both read a **TXT resources** file.

Scripts: - `Image_Features_Extraction_cif.py` (skimage) - `Pyfeats_Image_Features_Extraction_cif.py` (parallel)

Resources (TXT) — example

Create `resources_features.txt`:

```
# Required
images_path=/abs/path/to/Segmented_images_CellPose/Tiff_images
features_export_path=/abs/path/to/Features_Output
experiments_types=WT control experiment,Human normal donors

# Channel tags in segmented filenames
bf_tags=BF
nucleus_tags=Nucleus
df_tags=DF

# Optional extras
# additional_roles=Actin
# Actin_tags=Actin

# Skimage properties (optional; defaults used if omitted)
#
properties_list=area,eccentricity,equivalent_diameter_area,extent,perimeter,solidity,axis_majo

# Pyfeats backend & workers
pyfeats_backend=pyfeats_script:Pyfeats_Features_Extraction
n_workers=4
```

> **Backend function** for Pyfeats: provide an importable callable like `pyfeats_script.Pyfeats_Features_Extraction(img_uint8, mask_uint8) -> dict`. All returned keys are suffixed per role.

Run

```
python Image_Features_Extraction_cif.py /path/to/resources_features.txt
python Pyfeats_Image_Features_Extraction_cif.py /path/to/
resources_features.txt
```

Outputs (CSV names you asked for)

Written to `features_export_path/`:

```
skimage_<EXPERIMENT>_<SUB>.csv  # e.g., skimage_WT control
experiment_GFP_pos.csv
pyfeats_<EXPERIMENT>_<SUB>.csv  # e.g., pyfeats_WT control
experiment_GFP_neg.csv
```

- `Image_Type` column is `<EXPERIMENT>/<SUB>`; `Image_Name` is the set base.
- Feature columns are suffixed with `_BF`, `_Nucleus`, `_DF` (and any additional roles you define).
- Files are grouped by **base set** (same name except channel token).

---

3) Prediction (XGBoost)

Script: `prediction.py`
(renamed from `predict_with_renaming.py` *as requested)*

What it does

- Reads a skimage features CSV.
- **Normalizes** legacy columns to the new convention:
- `feature` → `feature_BF`
- `feature.1` → `feature_Nucleus`
- `feature.2` → `feature_DF`
- Aligns the DataFrame columns to the **model's training order** (from the checkpoint; or from a `feature_names.txt`).
- Predicts and writes two JSON files:
- `<features>_proba.json` – per-row probability vector
- `<features>_pred.json` – per-row predicted class

Usage

```
python prediction.py
  --features "/path/to/skimage_WT control experiment_GFP_pos.csv"
  --model "/path/to/xgb_checkpoint.pkl"
  --feature-list "/path/to/feature_names.txt" \  # only if model has no names
  --output-dir "/path/to/out"
  --id-cols "Image_Type,Image_Name"
```

Notes

- Supports models saved as **sklearn** (`joblib` / `pickle`) or raw **xgboost.Booster** (`.json/.ubj/.model`).
- If the checkpoint doesn't carry feature names, supply `--feature-list` with names in the **exact training order** (one per line or a CSV header).
- The script also writes a `*_normalized.csv` (new-style column names) for auditability.

Naming Conventions & Sets

- Channels are identified in filenames using the configured **tags** and then **standardized** to role names: `BF`, `Nucleus`, `DF`, plus optional `ADDITIONAL_ROLES`.
- An **image set** = files that differ only by the channel token (e.g., `11_BF*`, `11_Nucleus*`, `11_DF*`).
- **Segmentation** requires all **declared core roles** per set (those you listed in the segmentation resources TXT). If any are missing, the set is **skipped** and reported in `channels_unavailability_<k>.txt` (or dry run report).

Reproducibility & Re-runs

- Segmentation, planning, and logs are **versioned** by an auto-incrementing run index (`_0`, `_1`, …).
  Previously completed sets are recognized and **not** reprocessed.
- Feature extraction is stateless; you can re-run at any time—files are re-written.
- Prediction is stateless; re-run as needed.

Troubleshooting

- **Segmentation finds no images**: check `path_for_raw_images` and `experiments_types` in the segmentation resources TXT; verify there are `.tif/.tiff` files.
- **Sets are skipped**: see `channels_unavailability_<k>.txt`. A required core channel is missing for those sets.
- **Feature CSV has no rows**: ensure you pointed `images_path` to the **Tiff_images** directory and the experiment/subfolder names match.
- **Pyfeats is slow**: increase `n_workers` in the features TXT (bounded by CPU/memory).
- **Prediction errors about missing features**: provide `--feature-list` in the same order used during training; ensure your legacy columns are correctly mapped to `_BF/_Nucleus/_DF`.

Suggested Repo Layout

```
repo/
  segment_cells_refactored.py
  Image_Features_Extraction_cif.py
  Pyfeats_Image_Features_Extraction_cif.py
  prediction.py
  resources.txt                      # segmentation
  resources_features.txt             # feature extraction
  README.md
```

Keep your model checkpoints (e.g., `xgb_checkpoint.pkl`) outside of version control if they're large.

Acknowledgements

- Cellpose for segmentation
- scikit-image for regionprops
- XGBoost for classification

If you want CI commands, Dockerfile, or a `requirements.txt`, say the word and I'll add them.