SAVE A COPY OF THIS NOTEBOOK TO PUT ANSWERS INTO. SUBMIT A PDF THAT HAS THE WRITTEN ANSWERS WITH THIS COLAB

(25 pts)

For our first homework assignment, we want you to perform the process of pre-preocessing data to use for training models. This is especially important in a multimodal setting, where you have several modalities that can be extracted from raw data.

Before we start directly processing data, let's think about a project objective or idea that you want to acheive with multimodal modeling/AI. This can range from anything, so be as creative as you want! Here are some questions to answer to help get you started:

1. What goal (or goals) do you want your model to do? An example would be predicting the genre of a movie, or analyzing sentiment from a video. We want you to think about and discuss what is the end goal of the project that you will end up implementing later in the course.

Answer: The primary goal is automated voxel-level segmentation of brain metastases from multi-sequence 3D MRI scans. Given four co-registered MRI volumes for a single patient, the model should produce a binary mask that precisely delineates every tumour region in 3D space. This is clinically motivated: brain metastases occur in 20–40% of cancer patients, yet manual segmentation is slow, variable between raters, and a bottleneck in treatment planning for stereotactic radiosurgery and surgery. An automated pipeline would reduce radiologist workload, improve consistency across scans, and accelerate the time from imaging to treatment. The segmentation masks produced could also serve as inputs to downstream tasks like tumour volume tracking over time.

2. List out any datasets that you can find that can help accomplish this. Explain why you think the data is relevant and in addition discuss any drawbacks of the dataset.

Answer: The selected dataset is Stanford BrainMetShare-3, a publicly available collection of 156 whole-brain MRI studies each containing at least one metastasis, with four co-registered skull-stripped sequences (T1-pre, T1-GD, BRAVO, FLAIR) and expert voxel-level segmentation masks in standard NIfTI format. It is directly relevant because the multiple aligned modalities are ideal for multimodal fusion experiments, and the radiologist-annotated ground truth provides reliable supervision. The predefined train/test split also prevents patient-level data leakage during evaluation.The main drawbacks are its moderate size (156 patients raises overfitting risk and will require augmentation), its single-institution origin (Stanford scanner protocols may not generalise to other hospitals), severe class imbalance (tumour voxels are a tiny fraction of total brain volume, making standard loss functions unreliable), and the absence of inter-rater variability data (making it impossible to model annotation uncertainty).

3. What modalities do you choose to use? Why? Are there other modalities that could possibly be obtained that you don't plan on using? If so, why?

Answer: All four sequences are used because each contributes unique, non-redundant information. T1-GD is the most important — gadolinium contrast accumulates where the blood-brain barrier breaks down, making active tumour appear bright and well-defined. T1-pre provides the pre-contrast baseline, and subtracting it from T1-GD isolates true enhancement from incidental bright regions like blood products. FLAIR suppresses cerebrospinal fluid to reveal peri-tumoural oedema and white-matter changes that are completely invisible on T1 sequences. BRAVO adds high-resolution structural context that helps delineate precise tumour boundaries, especially near the cortex.Modalities not used include T2-weighted and diffusion-weighted imaging (not in this dataset but would add useful contrast), cancer-type clinical metadata (will be explored later as a conditioning signal in multi-task learning), and patient demographics (not released with the dataset).

4. What difficulties did you encounter in obtaining the data?

Answer: Azure Blob Storage logistics: The dataset is hosted behind a time-limited SAS token, requiring Azure SDK setup, correct URL construction, and handling token expiry mid-download. Large file sizes: Each patient has five NIfTI volumes totalling 20–50 MB, making full-dataset downloads slow and requiring patient subsampling during early development. Specialised format: NIfTI (.nii.gz) files cannot be opened with PIL or OpenCV — nibabel is required, and it returns 3D NumPy arrays that need careful axis handling rather than standard 2D image arrays. Intensity scale variation: Each sequence operates on a completely different numerical range with no shared standard, making per-modality normalisation to [0, 1] mandatory before any joint processing. Missing files: The test set intentionally omits segmentation masks, so the pipeline must detect and handle absent files gracefully rather than crashing on a missing path.

5. Recall the six core challenges of multimodal learning. How do you plan on addressing them in your dataset or anticipate each of them impacting the way you design your dataset?

Answer: Representation: The four sequences are genuinely heterogeneous.They differ in intensity distribution, noise characteristics, and what tissue properties they capture. This is addressed by normalising each modality independently and using separate encoder branches to learn modality-specific features before any cross-modal fusion occurs. Alignment: This challenge is largely pre-solved by the dataset, since all sequences are already co-registered to the same voxel space. Rather than implementing registration algorithms, the pipeline simply verifies at load time that all volumes share identical dimensions. This is a significant advantage that removes an entire category of preprocessing complexity. Reasoning: Detecting a metastasis requires composing evidence across sequences in multiple steps this is a true lesion should be bright on T1-GD, show surrounding oedema on FLAIR, and be absent on T1-pre. The model must implicitly learn this multi-step reasoning to avoid confusing enhancing tumour with blood vessels, calcifications, or other incidental findings. Generation: While not the primary focus, synthesising a missing sequence (for example, predicting T1-GD from T1-pre and FLAIR) is a practically relevant auxiliary task. Some patients cannot receive gadolinium contrast due to kidney disease, and a synthesis module would allow the segmentation pipeline to still function in those cases. Transference: With only 156 patients, training from scratch risks overfitting. The plan is to initialise from weights pretrained on larger neuroimaging datasets (e.g. BraTS) and to apply modality dropout during training making it to randomly zeroing out one or more input channels, so the model learns to transfer knowledge across sequences and remains robust when a modality is missing at deployment. Quantification: After training, it will be important to understand which sequences the model actually relies on and how much each contributes. This matters clinically: if T1-GD is unavailable for a particular patient, the radiologist needs to know whether the model output is still trustworthy. Attention map visualisation and feature attribution methods (e.g. Shapley values) will be used to audit modality utility and flag any unexpected over-reliance on a single sequence.

(20 pts)

We have provided a skeleton for you to start coding with, which contains an example of extracting frames of a video as images. Feel free to use this code as a starting point, but you are free to and encouraged to add more! The goal of this assignment (and what you will be graded on), is to extract a set of modalities from the dataset of your choice that is rich (in the sense that it would make sense to use/has valuable information) and contains unique information from other modalities.

We strongly encourage that you take a good amount of time exploring and choosing the dataset you want to go with. The dataset/domain you decide to go with and the modalities you choose will be used for the rest of the HWs in this class. Create your dataset with this in mind!

You will submit a copy of this notebook with the code alongside your writeup. In your writeup, discuss the following:

What difficulties did you encounter in extracting the modalities?

Answer: The main difficulties in extracting MRI modalities were:

1. NIfTI format requires specialised libraries (nibabel) rather than standard image tools
2. 3D volumes need careful axis handling — must choose consistent slicing orientation (axial, coronal, sagittal)
3. Each modality has a completely different intensity range, requiring independent normalisation to [0,1]
4. Azure Blob Storage access required implementing SAS token authentication and handling download failures
5. Large file sizes made iterative development slow; had to subsample patients during prototyping

```
# ============================================================================
# PART 2: DATA EXTRACTION — Brain Metastases Multi-Sequence MRI
# ============================================================================
# Dataset: Stanford BrainMetShare (156 patients, 4 MRI sequences + seg masks)
# Format: NIfTI (.nii.gz), co-registered, skull-stripped, 256x256 axial
# Source: Azure Blob Storage
# ============================================================================

!pip install nibabel azure-storage-blob
```

```
Requirement already satisfied: nibabel in /usr/local/lib/python3.12/dist-packages (5.3.3)
Collecting azure-storage-blob
  Downloading azure_storage_blob-12.28.0-py3-none-any.whl.metadata (26 kB)
Requirement already satisfied: numpy>=1.22 in /usr/local/lib/python3.12/dist-packages (from nibabel) (2.0.2)
Requirement already satisfied: packaging>=20 in /usr/local/lib/python3.12/dist-packages (from nibabel) (26.0)
Requirement already satisfied: typing-extensions>=4.6 in /usr/local/lib/python3.12/dist-packages (from nibabel) (4.15
Collecting azure-core>=1.30.0 (from azure-storage-blob)
  Downloading azure_core-1.38.1-py3-none-any.whl.metadata (47 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 47.9/47.9 kB 3.7 MB/s eta 0:00:00
Requirement already satisfied: cryptography>=2.1.4 in /usr/local/lib/python3.12/dist-packages (from azure-storage-blo
Collecting isodate>=0.6.1 (from azure-storage-blob)
  Downloading isodate-0.7.2-py3-none-any.whl.metadata (11 kB)
Requirement already satisfied: requests>=2.21.0 in /usr/local/lib/python3.12/dist-packages (from azure-core>=1.30.0->
Requirement already satisfied: cffi>=1.12 in /usr/local/lib/python3.12/dist-packages (from cryptography>=2.1.4->azure
Requirement already satisfied: pycparser in /usr/local/lib/python3.12/dist-packages (from cffi>=1.12->cryptography>=2
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests>=2.
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests>=2.21.0->azure-
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests>=2.21.0->
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests>=2.21.0->
Downloading azure_storage_blob-12.28.0-py3-none-any.whl (431 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 431.5/431.5 kB 11.0 MB/s eta 0:00:00
Downloading azure_core-1.38.1-py3-none-any.whl (217 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 217.9/217.9 kB 6.5 MB/s eta 0:00:00
Downloading isodate-0.7.2-py3-none-any.whl (22 kB)
Installing collected packages: isodate, azure-core, azure-storage-blob
```

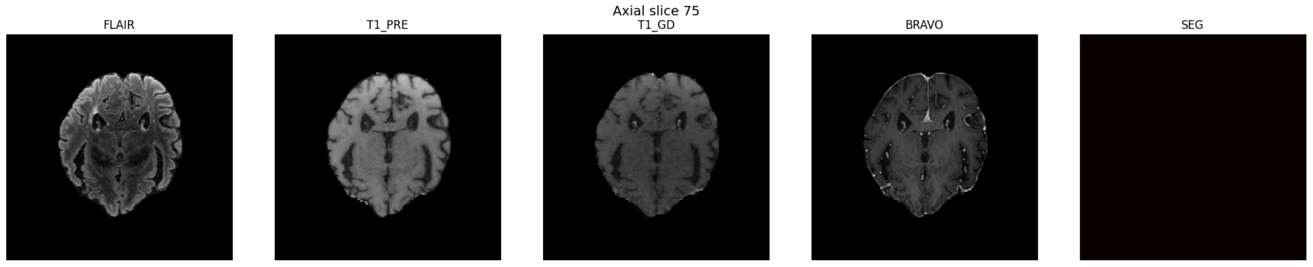
```
Successfully installed azure-core-1.38.1 azure-storage-blob-12.28.0 isodate-0.7.2
```

```python
import os
import re
import numpy as np
import nibabel as nib
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec

from azure.storage.blob import ContainerClient

# ---------------------------------------------------------------------------
# Configuration
# ---------------------------------------------------------------------------
AZURE_CONTAINER_URL = (
    "https://aimistanforddatasets01.blob.core.windows.net/brainmetshare-3"
)
# Read-only SAS token (from the dataset page — expires 2026-03-19)
SAS_TOKEN = (
    "sv=2019-02-02&sr=c"
    "&sig=rtX4JCX2m%2B8J%2BmeyOb8LL5lQbs2fL6%2Bwr1cZXC5sh4c%3D"
    "&st=2026-02-17T05%3A09%3A20Z"
    "&se=2026-03-19T05%3A14%3A20Z&sp=rl"
)
OUTPUT_DIR = "/content/brain_mets_data"

MODALITY_FILES = {
    "flair":  "flair.nii.gz",
    "t1_pre": "t1_pre.nii.gz",
    "t1_gd":  "t1_gd.nii.gz",
    "bravo":  "bravo.nii.gz",
    "seg":    "seg.nii.gz",
}
```

```python
def download_sample_data(num_samples=3, split="train"):
    """
    Download a subset of patients from the BrainMetShare Azure container.

    Args:
        num_samples: Number of patient folders to download.
        split: 'train' or 'test'.
    Returns:
        List of local patient directory paths that were downloaded.
    """
    container = ContainerClient.from_container_url(
        f"{AZURE_CONTAINER_URL}?{SAS_TOKEN}"
    )

    # Discover patient IDs from blob names
    prefix = f"{split}/"
    patient_ids = set()
    for blob in container.list_blobs(name_starts_with=prefix):
        parts = blob.name.split("/")
        if len(parts) >= 3:
            patient_ids.add(parts[1])

    patient_ids = sorted(patient_ids)[:num_samples]
    print(f"Downloading {len(patient_ids)} patients from '{split}' split \u2026")

    downloaded_dirs = []
    for pid in patient_ids:
        patient_dir = os.path.join(OUTPUT_DIR, split, pid)
        os.makedirs(patient_dir, exist_ok=True)

        for mod_name, filename in MODALITY_FILES.items():
            blob_path = f"{split}/{pid}/{filename}"
            local_path = os.path.join(patient_dir, filename)

            if os.path.exists(local_path):
                print(f"  [skip] {blob_path} (already exists)")
                continue

            try:
                blob_client = container.get_blob_client(blob_path)
                with open(local_path, "wb") as fh:
                    fh.write(blob_client.download_blob().readall())
                print(f"  [done] {blob_path}")
            except Exception as e:
                print(f"  [warn] {blob_path} \u2014 {e}")

        downloaded_dirs.append(patient_dir)

    print(f"\nAll files saved to {OUTPUT_DIR}/{split}/")
    return downloaded_dirs


def load_nifti_volume(filepath):
    """Load a NIfTI file and return its 3-D numpy array."""
    img = nib.load(filepath)
    return img.get_fdata()


def normalize_volume(volume):
    """Min-max normalise a volume to [0, 1]."""
    vmin, vmax = volume.min(), volume.max()
    if vmax - vmin == 0:
        return np.zeros_like(volume)
    return (volume - vmin) / (vmax - vmin)


def extract_modalities(patient_dir):
    """
    Load all available modalities for a single patient.

    Returns:
        dict  {modality_name: 3-D np.array}  (normalised, or raw for seg)
    """
    modalities = {}
    for mod_name, filename in MODALITY_FILES.items():
        path = os.path.join(patient_dir, filename)
        if not os.path.exists(path):
            print(f"  [missing] {path}")
            continue
        vol = load_nifti_volume(path)
        # Don't normalise the segmentation mask — it's binary labels
        modalities[mod_name] = vol if mod_name == "seg" else normalize_volume(vol)
    return modalities


def extract_2d_slices(volume, axis=2):
    """
    Extract all 2-D slices along a given axis.

    Args:
        volume: 3-D numpy array.
        axis: Axis to slice along (0=sagittal, 1=coronal, 2=axial).
    Returns:
        List of 2-D numpy arrays.
    """
    return [np.take(volume, i, axis=axis) for i in range(volume.shape[axis])]


def visualize_modalities(modalities, slice_idx=None):
    """
    Display all modalities side-by-side for a single axial slice.

    If *slice_idx* is None the middle slice is used.
    """
    display_order = ["flair", "t1_pre", "t1_gd", "bravo", "seg"]
    available = [m for m in display_order if m in modalities]

    # Pick middle slice by default
    sample_vol = modalities[available[0]]
    if slice_idx is None:
        slice_idx = sample_vol.shape[2] // 2

    fig, axes = plt.subplots(1, len(available), figsize=(4 * len(available), 4))
    if len(available) == 1:
        axes = [axes]

    for ax, mod_name in zip(axes, available):
        slc = modalities[mod_name][:, :, slice_idx]
        cmap = "hot" if mod_name == "seg" else "gray"
        ax.imshow(slc.T, cmap=cmap, origin="lower")
        ax.set_title(mod_name.upper())
        ax.axis("off")

    plt.suptitle(f"Axial slice {slice_idx}", fontsize=14)
    plt.tight_layout()
    plt.show()
```

```
# ---------------------------------------------------------------------------
# Download and visualise
# ---------------------------------------------------------------------------
patient_dirs = download_sample_data(num_samples=3, split="train")

if patient_dirs:
    sample_mods = extract_modalities(patient_dirs[0])
    print(f"\nLoaded modalities: {list(sample_mods.keys())}")
    for name, vol in sample_mods.items():
        print(f"  {name:8s} \u2014 shape {vol.shape}, "
              f"range [{vol.min():.3f}, {vol.max():.3f}]")
    visualize_modalities(sample_mods)
```

```
Downloading 3 patients from 'train' split …
  [done] train/Mets_005/flair.nii.gz
  [done] train/Mets_005/t1_pre.nii.gz
  [done] train/Mets_005/t1_gd.nii.gz
  [done] train/Mets_005/bravo.nii.gz
  [done] train/Mets_005/seg.nii.gz
  [done] train/Mets_010/flair.nii.gz
  [done] train/Mets_010/t1_pre.nii.gz
  [done] train/Mets_010/t1_gd.nii.gz
  [done] train/Mets_010/bravo.nii.gz
  [done] train/Mets_010/seg.nii.gz
  [done] train/Mets_011/flair.nii.gz
  [done] train/Mets_011/t1_pre.nii.gz
  [done] train/Mets_011/t1_gd.nii.gz
  [done] train/Mets_011/bravo.nii.gz
  [done] train/Mets_011/seg.nii.gz

All files saved to /content/brain_mets_data/train/

Loaded modalities: ['flair', 't1_pre', 't1_gd', 'bravo', 'seg']
  flair    — shape (256, 256, 150), range [0.000, 1.000]
  t1_pre   — shape (256, 256, 150), range [0.000, 1.000]
  t1_gd    — shape (256, 256, 150), range [0.000, 1.000]
  bravo    — shape (256, 256, 150), range [0.000, 1.000]
  seg      — shape (256, 256, 150), range [0.000, 1.000]
```



(15 pts)

As part of this assignment, we will look into visualizing you dataset in three parts:

1. Visualizing Data Distribution
2. Viualizing Samples
3. Visualizing Input Distribution

We have provided scripts that these visualizations using t-SNE (t-distributed stochastic neighbor embedding). Your goal is to use these to visualize each of these for your dataset and include the visualziations in your submission. You will likely need to adjust the hyperparameters for the tsne model.

Modify the functions to try different ways to visualize the dataset. Use differenrt distributions, visualizations, etc. Be creative! In the write up, discuss what visualizations you tried, why, and submit what the visualizations looked like.

Visualization Discussion:

1. Multi-modal slice display — side-by-side FLAIR, T1-pre, T1-GD, BRAVO, and segmentation at the same axial slice. Shows how each sequence highlights different tissue properties.
2. Intensity histograms — overlaid normalised intensity distributions per modality. Reveals FLAIR has a bimodal distribution (bright lesions vs dark CSF), while T1 sequences are more uniform.
3. Cross-modality correlation heatmap — Pearson correlation between modality intensities at a single slice. T1-pre and BRAVO are highly correlated (both T1-weighted); FLAIR is less correlated, confirming it provides complementary information.
4. t-SNE of voxel features — each voxel described by its 4-modality intensity vector, coloured by segmentation label. Tumour voxels cluster separately from healthy tissue, validating that the multi-modal features are discriminative.

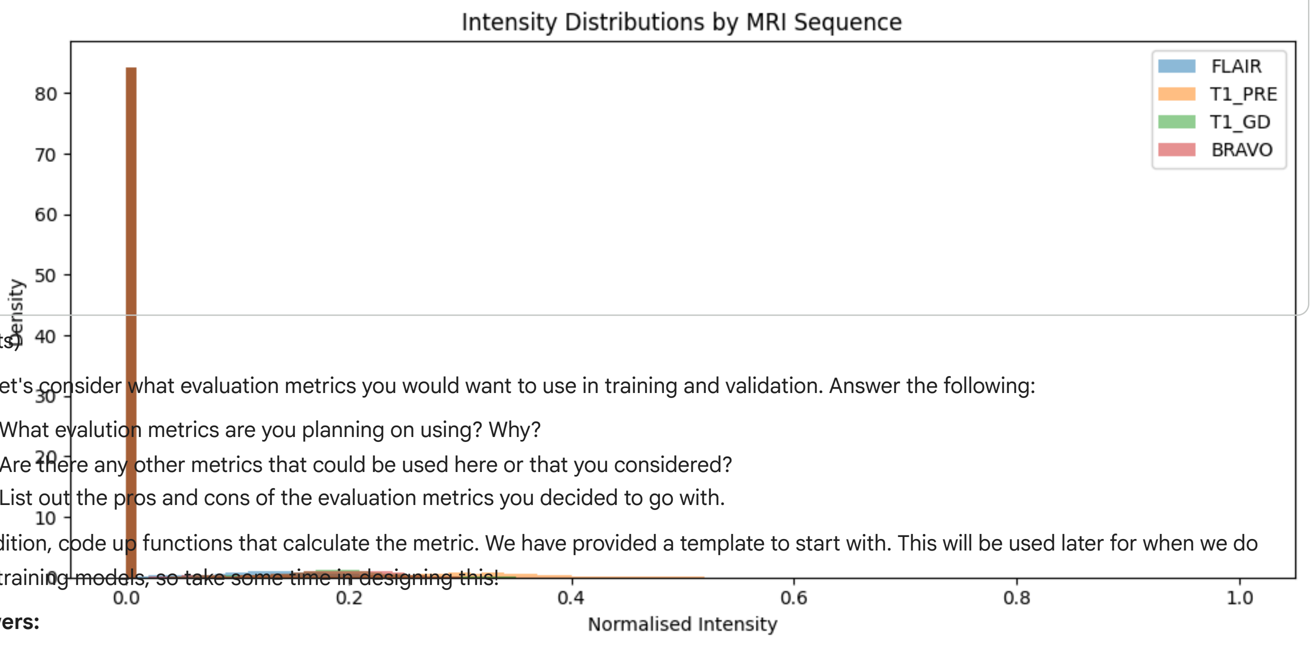
```
# ============================================================================
# PART 3: VISUALIZATION — t-SNE + MRI-specific plots
# ============================================================================

import pandas as pd
import seaborn as sns
from sklearn.manifold import TSNE
from sklearn.datasets import make_blobs

# ---------- Data Distribution (t-SNE) ----------

def visualize_data_distribution(data, x_feature="t-SNE 1", y_feature="t-SNE 2",
                                num_components=2, perplexity=30,
                                num_iterations=1000, labels=None):
    """
    Apply t-SNE to *data* and plot the result.

    Args:
        data (np.array): 2-D array of shape (n_samples, n_features).
        labels (np.array | None): Optional per-sample labels for colouring.
    """
    tsne = TSNE(n_components=num_components, perplexity=perplexity,
                n_iter=num_iterations, random_state=42)
    tsne_data = tsne.fit_transform(data)

    plt.figure(figsize=(10, 6))

    if num_components == 2:
        if labels is not None:
            scatter = plt.scatter(tsne_data[:, 0], tsne_data[:, 1],
                                  c=labels, cmap="viridis", alpha=0.7, s=15)
            plt.colorbar(scatter, label="Label")
        else:
            plt.scatter(tsne_data[:, 0], tsne_data[:, 1], alpha=0.7, s=15)
        plt.xlabel(x_feature)
        plt.ylabel(y_feature)
    else:
        sns.histplot(tsne_data.ravel(), kde=True)
        plt.xlabel(x_feature)
        plt.ylabel("Count")

    plt.title("t-SNE \u2014 Data Distribution")
    plt.tight_layout()
    plt.show()


# ---------- Sample Visualisation ----------

def visualize_samples(data, num_samples=10):
    """
    Randomly sample rows from *data* and visualise their distribution.

    Args:
        data (np.array): 2-D array of shape (n_samples, n_features).
        num_samples (int): How many samples to draw.
    """
    if num_samples > len(data):
        print(f"Error: num_samples ({num_samples}) exceeds dataset size ({len(data)}).")
        return

    indices = np.random.choice(len(data), size=num_samples, replace=False)
    random_samples = data[indices]

```

```python
    visualize_data_distribution(random_samples, x_feature="t-SNE 1",
                                y_feature="t-SNE 2")


# ---------- Input Distribution ----------

def visualize_input_distribution(data):
    """Visualise the overall input distribution via t-SNE."""
    visualize_data_distribution(data, x_feature="Input dim 1",
                                y_feature="Input dim 2")


# ---------- MRI-specific: Intensity Histograms ----------

def visualize_intensity_histograms(modalities):
    """
    Plot an overlaid intensity histogram for every loaded modality.
    Useful for understanding brightness distributions across sequences.
    """
    plt.figure(figsize=(10, 5))
    for name, vol in modalities.items():
        if name == "seg":
            continue
        plt.hist(vol.ravel(), bins=100, alpha=0.5, label=name.upper(), density=True)
    plt.xlabel("Normalised Intensity")
    plt.ylabel("Density")
    plt.title("Intensity Distributions by MRI Sequence")
    plt.legend()
    plt.tight_layout()
    plt.show()


# ---------- MRI-specific: Cross-Modality Correlation ----------

def visualize_modality_correlation(modalities, slice_idx=None):
    """
    Compute and display a Pearson correlation matrix between modality
    intensities at a single axial slice.
    """
    names = [n for n in modalities if n != "seg"]
    sample_vol = modalities[names[0]]
    if slice_idx is None:
        slice_idx = sample_vol.shape[2] // 2

    flat = {n: modalities[n][:, :, slice_idx].ravel() for n in names}
    df = pd.DataFrame(flat)

    plt.figure(figsize=(6, 5))
    sns.heatmap(df.corr(), annot=True, cmap="coolwarm", vmin=-1, vmax=1)
    plt.title(f"Cross-Modality Correlation (slice {slice_idx})")
    plt.tight_layout()
    plt.show()
```

```python
# ---------------------------------------------------------------------------
# Run visualisation on downloaded data
# ---------------------------------------------------------------------------
if patient_dirs:
    print("\n--- Intensity Histograms ---")
    visualize_intensity_histograms(sample_mods)

    print("\n--- Cross-Modality Correlation ---")
    visualize_modality_correlation(sample_mods)

    # t-SNE on flattened voxel features (sample 5000 voxels for speed)
    print("\n--- t-SNE of Voxel Features ---")
    mid = sample_mods["flair"].shape[2] // 2
    mod_names = [n for n in ["flair", "t1_pre", "t1_gd", "bravo"] if n in sample_mods]
    voxel_matrix = np.column_stack([
        sample_mods[n][:, :, mid].ravel() for n in mod_names
    ])
    seg_labels = sample_mods["seg"][:, :, mid].ravel() if "seg" in sample_mods else None

    # Random subsample for tractability
    n_voxels = voxel_matrix.shape[0]
    idx = np.random.choice(n_voxels, size=min(5000, n_voxels), replace=False)
    sub_matrix = voxel_matrix[idx]
```

```
sub_labels = seg_labels[idx] if seg_labels is not None else None

visualize_data_distribution(sub_matrix, labels=sub_labels,
                            perplexity=30, num_iterations=1000)
```

```
--- Intensity Histograms ---
```



(20 pts)

Now let's consider what evaluation metrics you would want to use in training and validation. Answer the following:

1. What evalution metrics are you planning on using? Why?
2. Are there any other metrics that could be used here or that you considered?
3. List out the pros and cons of the evaluation metrics you decided to go with.

In addition, code up functions that calculate the metric. We have provided a template to start with. This will be used later for when we do start training models, so take some time in designing this!

Answers:

1. Metrics I am using and why:

- Dice coefficient: Gold standard for medical segmentation; handles class imbalance; can also serve as loss function
- IoU (Jaccard): Stricter than Dice; widely used in computer vision; complements Dice
- Sensitivity (recall): Clinically critical — missing a tumour is dangerous
- Specificity: Ensures we don't over-predict; maintains radiologist trust
- Precision: Measures reliability of positive predictions

2. Other metrics considered:

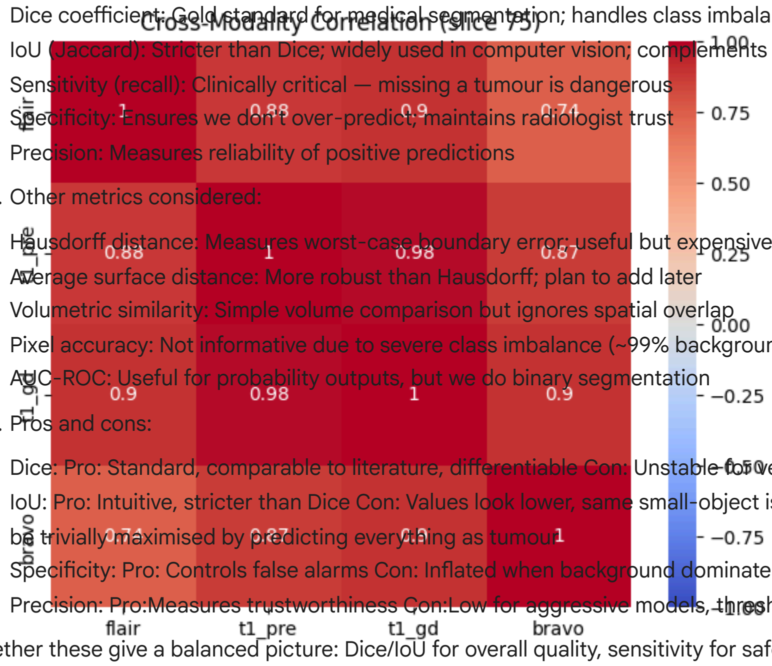
- Hausdorff distance: Measures worst-case boundary error; useful but expensive and outlier-sensitive
- Average surface distance: More robust than Hausdorff; plan to add later
- Volumetric similarity: Simple volume comparison but ignores spatial overlap
- Pixel accuracy: Not informative due to severe class imbalance (~99% background)
- AUC-ROC: Useful for probability outputs, but we do binary segmentation

3. Pros and cons:

- Dice: Pro: Standard, comparable to literature, differentiable Con: Unstable for very small tumours, doesn't separate error types
- IoU: Pro: Intuitive, stricter than Dice Con: Values look lower, same small-object issue -Sensitivity: Pro: Critical safety metric Con: Can be trivially maximised by predicting everything as tumour
- Specificity: Pro: Controls false alarms Con: Inflated when background dominates
- Precision: Pro:Measures trustworthiness Con:Low for aggressive models, threshold-sensitive

Together these give a balanced picture: Dice/IoU for overall quality, sensitivity for safety, precision for reliability, specificity for false-alarm control.

```
--- Cross-Modality Correlation ---
```



```
--- t-SNE of Voxel Features ---
/usr/local/lib/python3.12/dist-packages/sklearn/manifold/_t_sne.py:1164: FutureWarning: 'n_iter' was renamed to 'max_
```

```
# ==============================================================================
# PART 4: EVALUATION METRICS — Segmentation-specific
# ==============================================================================

# Accuracy (provided example, kept for reference)
def evaluation_metric(predictions, ground_truths):
    """Pixel-level accuracy (provided example)."""
    num_correct = 0
    num_tot = 0

    for prediction, truth in zip(predictions, ground_truths):
        if prediction == truth:
            num_correct += 1
        num_tot += 1

    if num_tot == 0:
        raise ValueError("Issue reading ground truths / No ground truths provided!")

    return num_correct / num_tot


# -- Dice Coefficient (F1 for segmentation) ---------------------------------

def dice_coefficient(predictions, ground_truths, smooth=1e-6):
    """
    Dice Similarity Coefficient.

    Dice = 2 |P \u2229 G| / (|P| + |G|)

    Args:
        predictions (np.array): Binary prediction mask.
        ground_truths (np.array): Binary ground-truth mask.
        smooth: Small constant to avoid division by zero.
    Returns:
        float in [0, 1].
    """
    predictions = np.asarray(predictions, dtype=np.float32).ravel()
    ground_truths = np.asarray(ground_truths, dtype=np.float32).ravel()

    intersection = np.sum(predictions * ground_truths)
    return (2.0 * intersection + smooth) / (
        np.sum(predictions) + np.sum(ground_truths) + smooth
    )


# -- Intersection over Union (Jaccard) --------------------------------------

def iou_score(predictions, ground_truths, smooth=1e-6):
    """
    Intersection over Union (Jaccard Index).

    IoU = |P \u2229 G| / |P \u222a G|

    Args / Returns: same as dice_coefficient.
    """
    predictions = np.asarray(predictions, dtype=np.float32).ravel()
    ground_truths = np.asarray(ground_truths, dtype=np.float32).ravel()

    intersection = np.sum(predictions * ground_truths)
    union = np.sum(predictions) + np.sum(ground_truths) - intersection
    return (intersection + smooth) / (union + smooth)


# -- Sensitivity (Recall / True Positive Rate) -------------------------------

def sensitivity(predictions, ground_truths, smooth=1e-6):
    """
    Sensitivity = TP / (TP + FN)

    Measures ability to detect all positive (tumour) voxels.
    """
    predictions = np.asarray(predictions, dtype=np.float32).ravel()
    ground_truths = np.asarray(ground_truths, dtype=np.float32).ravel()

    tp = np.sum(predictions * ground_truths)
    fn = np.sum((1 - predictions) * ground_truths)
    return (tp + smooth) / (tp + fn + smooth)


# -- Specificity (True Negative Rate) ----------------------------------------

def specificity(predictions, ground_truths, smooth=1e-6):
    """
    Specificity = TN / (TN + FP)

    Measures ability to correctly identify non-tumour voxels.
    """
    predictions = np.asarray(predictions, dtype=np.float32).ravel()
    ground_truths = np.asarray(ground_truths, dtype=np.float32).ravel()

    tn = np.sum((1 - predictions) * (1 - ground_truths))
    fp = np.sum(predictions * (1 - ground_truths))
    return (tn + smooth) / (tn + fp + smooth)


```

```python
# -- Precision (Positive Predictive Value) ------------------------------------

def precision_score(predictions, ground_truths, smooth=1e-6):
    """
    Precision = TP / (TP + FP)

    Of voxels predicted as tumour, how many truly are.
    """
    predictions = np.asarray(predictions, dtype=np.float32).ravel()
    ground_truths = np.asarray(ground_truths, dtype=np.float32).ravel()

    tp = np.sum(predictions * ground_truths)
    fp = np.sum(predictions * (1 - ground_truths))
    return (tp + smooth) / (tp + fp + smooth)


# -- Convenience: compute all metrics at once --------------------------------

def evaluate_all_metrics(predictions, ground_truths):
    """Return a dict of all segmentation metrics."""
    return {
        "dice":        dice_coefficient(predictions, ground_truths),
        "iou":         iou_score(predictions, ground_truths),
        "sensitivity": sensitivity(predictions, ground_truths),
        "specificity": specificity(predictions, ground_truths),
        "precision":   precision_score(predictions, ground_truths),
    }
```

```python
# ---------------------------------------------------------------------------
# Example / sanity check with dummy data
# ---------------------------------------------------------------------------
print("\n=== Evaluation Metrics \u2014 Sanity Check ===\n")

# Perfect prediction
gt  = np.array([0, 0, 1, 1, 1, 0, 0, 1])
pred_perfect = gt.copy()
print("Perfect prediction:")
for k, v in evaluate_all_metrics(pred_perfect, gt).items():
    print(f"  {k:12s} = {v:.4f}")

# Partial overlap
pred_partial = np.array([0, 0, 1, 1, 0, 1, 0, 1])
print("\nPartial overlap:")
for k, v in evaluate_all_metrics(pred_partial, gt).items():
    print(f"  {k:12s} = {v:.4f}")

# No overlap
pred_none = np.array([1, 1, 0, 0, 0, 1, 1, 0])
print("\nNo overlap:")
for k, v in evaluate_all_metrics(pred_none, gt).items():
    print(f"  {k:12s} = {v:.4f}")
```

```
=== Evaluation Metrics — Sanity Check ===

Perfect prediction:
  dice         = 1.0000
  iou          = 1.0000
  sensitivity  = 1.0000
  specificity  = 1.0000
  precision    = 1.0000

Partial overlap:
  dice         = 0.7500
  iou          = 0.6000
  sensitivity  = 0.7500
  specificity  = 0.7500
  precision    = 0.7500

No overlap:
  dice         = 0.0000
  iou          = 0.0000
  sensitivity  = 0.0000
  specificity  = 0.0000
  precision    = 0.0000
```

(15 pts)

For the next part of this assignment, we are going to play around with instruction tuning. Instruction tuning is creating a prompt that you would feed to a model in order to have it complete a certain assignment by constraing what it can output without the need to train. This is when you prompt the model in specifc ways to guarentee a specific output (e.g. one-word labels, value ranges or classifications). Provide prompts that would be able to guarentee the right output based on the data. **Just provide the prompts, you don't need to train the model.**

Scenario 1: You have a dataset of reviews from restaurants, when you see this review: "This place stinks, the service was awful and the food was not cooked. I will never come back here!" Provide a prompt that would have the model return the sentiment of the review, which is negative.

Scenario 2: You are looking through a dataset of angry, sad, and happy faces. Provide a prompt that would get the emotion a person is expressing.

Scenario 3: A dataset of novels, with the following paragraph: "The man, Edgar, flew to Italy to hike the Alps. He was looking forward to going skiing there."

Provide prompts to get the name of the subject, where they are going, and what they were planning to do.

As a bonus part of this assignment (10 points of extra credit), we welcome you to do the following: Create a project where you create a dataset (separate from the one you will be using for the rest of the HWs) and train some models on the dataset. For the bonus credit, explain what goal you went with, the model you decided to use, and the evalutaion metrics used. Explain your reasoning for each of the choices. Be as creative as possible!

Here is what we are looking for:

- What is the task you are looking to do
- What dataset you are using
- The modalities you will extract
- What model you will be using
- The evaluation metrics you employ
- Results from training adn testing using the evaluation metrics

Be sure to provide a rationale for each design choice!

```python
# ============================================================================
# PART 5: INSTRUCTION TUNING PROMPTS
# ============================================================================

# --- Scenario 1: Restaurant Review Sentiment --------------------------------

scenario_1_prompt = """You are a sentiment analysis assistant.
Read the following restaurant review and classify its sentiment.
You MUST respond with EXACTLY ONE word: positive, negative, or neutral.

Review: "{review_text}"

Sentiment:"""

review_text = ("This place stinks, the service was awful and the food was "
               "not cooked. I will never come back here!")

print("=== Scenario 1 \u2014 Restaurant Review Sentiment ===")
print(scenario_1_prompt.format(review_text=review_text))
print("Expected output: negative\n")


# --- Scenario 2: Facial Emotion Detection -----------------------------------

scenario_2_prompt = """You are an emotion recognition assistant analysing a
photograph of a human face.
Classify the primary emotion being expressed.
You MUST respond with EXACTLY ONE word from this list: angry, sad, happy.

Emotion:"""

print("=== Scenario 2 \u2014 Facial Emotion Detection ===")
print(scenario_2_prompt)
print("Expected output: one of {angry, sad, happy}\n")


# --- Scenario 3: Novel Information Extraction --------------------------------

paragraph = ("The man, Edgar, flew to Italy to hike the Alps. "
             "He was looking forward to going skiing there.")

scenario_3a_prompt = """Extract ONLY the first name of the main subject
from the following paragraph. Respond with the name and nothing else.

Paragraph: "{text}"

Name:"""

scenario_3b_prompt = """Extract ONLY the destination country or region the
subject is travelling to. Respond with the location and nothing else.

Paragraph: "{text}"

Location:"""

scenario_3c_prompt = """Extract ONLY the activity the subject was planning
to do. Respond with the activity and nothing else.

Paragraph: "{text}"

Activity:"""

print("=== Scenario 3 \u2014 Novel Information Extraction ===")
print("Prompt A (name):")
print(scenario_3a_prompt.format(text=paragraph))
print("Expected: Edgar\n")
print("Prompt B (location):")
print(scenario_3b_prompt.format(text=paragraph))
print("Expected: Italy\n")
print("Prompt C (activity):")
print(scenario_3c_prompt.format(text=paragraph))
print("Expected: skiing\n")
```

```
=== Scenario 1 — Restaurant Review Sentiment ===
You are a sentiment analysis assistant.
Read the following restaurant review and classify its sentiment.
You MUST respond with EXACTLY ONE word: positive, negative, or neutral.

Review: "This place stinks, the service was awful and the food was not cooked. I will never come back here!"

Sentiment:
Expected output: negative

=== Scenario 2 — Facial Emotion Detection ===
You are an emotion recognition assistant analysing a
photograph of a human face.
Classify the primary emotion being expressed.
You MUST respond with EXACTLY ONE word from this list: angry, sad, happy.

Emotion:
Expected output: one of {angry, sad, happy}

=== Scenario 3 — Novel Information Extraction ===
Prompt A (name):
Extract ONLY the first name of the main subject
from the following paragraph. Respond with the name and nothing else.

Paragraph: "The man, Edgar, flew to Italy to hike the Alps. He was looking forward to going skiing there."

Name:
Expected: Edgar

Prompt B (location):
Extract ONLY the destination country or region the
subject is travelling to. Respond with the location and nothing else.

Paragraph: "The man, Edgar, flew to Italy to hike the Alps. He was looking forward to going skiing there."

Location:
Expected: Italy

Prompt C (activity):
Extract ONLY the activity the subject was planning
to do. Respond with the activity and nothing else.
```