chaimeleon_open_challenges_rcjh - Lung Overall Survival

Algorithm: LungCombinedResnet18PretrainedModel (400 buckets 0.5 months per bucket)

Run inside of `./library/process_lung.py` this algorithm is a deep learning algorithm combining a residual block inspired metadata fed set of layers with a Resnet18 based image feature extractor trained to predict which bucket a survival event will occur in. Images fed into this model have 9 layers - three for each anatomical plane in the scan - but the Resnet18 based image feature extractor is built to take in only three layers. The combination of the output of the metadata layers with the input images acts to allow the metadata layers to highlight/enhance the input images while also reducing them to three layers - one for each anatomical plane. The metadata layers output nine values - one for each slice - and after these values are used to scale the input slices the slices along an anatomical plane are averaged. This theoretically allows different planes or slices to be weighted more highly in patients with specific clinical information patterns.

Models were trained with both a limited subset of clinical information (gender, smoking_status) and with all metadata values. Additionally, models were also trained with and without censorship by event type.

During training ground truth survival times were converted into buckets with a set number of months as their width. This made the problem a more traditional classification problem and models were trained using multi-label soft margin loss and Adam optimization. Early experiments with from scratch models showed higher levels of convergence than using mean squared error loss, however, the team had less time available for this particular challenge and was unable to test the efficacy of different loss functions for the model structures chosen for the release models.

Resnet18 was chosen as the basis pretrained model due to its relatively small size and the difficulties faced with fully from scratch models generalizing well. With the limited amount of data provided for training augmentation was extensively made use of with random affine transformations applied to the image and random scaling +/- 15% applied to the numerical metadata values and a small (0-5% chance) of categorical clinical information having their values switched.

Many of the choices made for the lung challenge were informed by learning from the prostate challenge and may not have been ideal. It would have been beneficial to investigate using MSE loss with the selected architecture among other things. Evaluation for this challenge in particular seemed hampered by the choice to use augmented validation sets as test sets rather than the existence of a true test set for local evaluation. However, with only ~300 cases available for both training and evaluation, it did not seem feasible to split the cases into three sets and still have enough data to create a generalized algorithm.

Team rcjh:

- Jordan Hendriksen

Methods - Classification Phase

Data Preparation

Common methods:

Both cancer types make use of a common parent class `ChaimeleonData`. This class encapsulates several methods used for reading and preparing data for use in training and evaluation.

- `process_data_directory`: Responsible for loading the raw data into the dataset. It uses the add_raw_case_data method to add the raw data to the dataset. This method looks for case folders in the specified data directory and adds the raw data for each case folder.
  - `add_raw_case_data`: Loads the image and metadata files for a given case and adds them to the dataset class. If the case_image_archive parameter is specified, the code loads the image data from the archive. If the release flag is set to True, it loads the data from the directory regardless of archive existence.
  - `add_raw_metadata` and `add_raw_ground_truth` methods simply open their respective json files and store the data within the file as dictionaries.
  - `load_image_file`: loads the image files (`.nii.gz` or `.mha`) into numpy arrays and applies a set of preprocessing (to be expanded upon below in the 'Prostate Risk Specific' and 'Lung Overall Survival Specific' sections below) to the image before adding them to the dataset class.
- `get_dataset_splits`: randomly splits the keys for the different cases within the dataset into train, validation, and test sets. A random seed is set - and can be defined at initialization of the dataset - while the percentage of cases targeted for each split is defined when the method is called.

After a dataset is initialized the first time it's images are stored as a compressed numpy array archive in 'npzc' format. During subsequent initializations if the release flag is not set to True the code loads the data from the archive if the archive still exists. This drastically cuts down on IO and computation if no parameters affecting the images loaded are being experimented with.

Lung Overall Survival Specific:

- `load_image_file` preprocessing for lung cases performs three main preprocessing steps:
  - aggregating the scans into a set of 3 'average' slices for each of the anatomical planes - resulting in 9 total slices that are concatenated into one array for use in training/evaluation
  - cropping 32 pixels off each edge of the initial tensor
  - scaling the values of the 'average' slices between 0 and 1
- Dataset preparation is completed in the `LungCancerDataset` class by doing the following for cases in the current metadata split:
  - normalizing metadata (one hot encoding categorical metadata) and loading both encoded categorical and numerical metadata into a vector. This was structured as [encoded categorical metadata (which values and order decided at runtime), numerical metadata]
  - normalizing ground truth values into an *N* value classification format. At initialization *N* is specified as the number of buckets the ground truth values will fall into with the number of months per bucket as specified. True ground truth values are divided by the number of months per bucket and the result is rounded - with that becoming the index of the buckets where the ground truth is then labeled at.
  - Image and metadata augmentation strategies are defined
    - only during train and 'test' - which for this phase was an augmented version of the validation set

Models and Training

Lung Overall Survival

- Training:
    - MultiLabel soft margin loss (used with training for submitted models)
    - MSE loss (used during experiments early on)
    - Tracked a variety of statistics and saved models based on both accuracy (number of validation samples with the correct classification) and score (concordance index)
    - Tracked variables required in order to recreate or *very nearly* recreate a model by using the same training parameters
- Models:
    - From scratch image model
    - From scratch metadata model
    - From scratch combined image metadata model
    - Combined image and metadata models from Resnet18 base (trained with various resnet layers frozen)
        - `LungCombinedResnet18PretrainedModel` - used for all three validation submissions and the test submission
            - 400 buckets with 0.5 months per bucket (v1 and v3) - submitted to test set - no resnet layers frozen
            - 50 buckets with 3 months per bucket (v2) - resnet layers 2 and 3 frozen
            - 400 buckets with 0.5 months per bucket (v3) - resnet layers 2 and 3 frozen

Evaluation

The actual evaluations were run inside a set of jupyter notebooks - `/notebooks/evaluate_prostate_models_V2.ipynb` and `/notebooks/evaluate_lung_models.ipynb`. Evaluations were performed by generating scores for each model on three datasets - the validation set used during training and two test sets.

In this phase due to a small number of cases in each cancers dataset the test sets were generated by applying various strengths of translation, rotation, skew, and scaling augmentations to the validation set to create images that the model 'had not' directly seen before. There are flaws in this system but it's the one that was chosen for the classification phase.

After generating scores for each model on each of the three datasets the scores were ranked and an average rank was calculated. Models were ordered by average rank and their individual scores against the different sets were compared to one another to try to determine which model might perform the best on the Grand Challenge datasets. I.e. a model with the best average rank because it did best on the validation set but faired more poorly on the test sets compared to the second or third average ranked model might not be the best choice as it seems to generalize more poorly.