Provenance information for digital pathology machine learning pipeline

This document contains a description of a machine learning pipeline to detect carcinogenic cells at prostate scans. The pipeline is developed at the Faculty of Informatics, Masaryk University.

Input data format
Input data set consists of original scans of slides (1 scan = 1 slide), as produced by a microscope and an annotation file prepared manually by a doctor. Each scan file is stored as an .mrxs file (kind of an index) and data files referenced by the mrxs. Annotations are stored as an xml. Each scan is identified by its ID, which is used as a part of a naming convention for scan and annotation files. Testing and training scans are determined using a name convention for corresponding files (a prefix). This input data set will be referenced as a raw dataset in this text.

Pipeline description

The machine learning pipeline (MLP) processes annotated images of slides and learns to detect the presence of carcinoma. The pipeline is implemented as a set of python scripts and is executed on a server (Metacentrum). The input dataset is stored on a filesystem, where it is accessed by the scripts.

The pipeline can be divided into the following three parts:

1. **Preprocessing**. Goal of the first part is to split the input images into tiles, in order to make it usable for machine learning based processing in the consecutive steps. The preprocessing algorithm firstly computes background and annotation masks for each scan[1]. Then a gz-table is computed for each scan, representing how the scan is split to tiles. Each record in a gz-table consists of the x and y coordinates of a particular tile (tile coordinates in the original scan) and a label (true/false, depending on whether the particular tile contains carcinogenic cells). Size of a tile is determined in a configuration file. All the gz-tables are then indexed in a HDF5 index[2].
2. **Training and testing**.
    a. This part processes a subset of the raw dataset (scans and their annotations) and the HDF5 table in order to train the machine learning model to detect carcinogen cells. Result of this step is a train_output directory with stored weights of the trained net (so that the result is a trained model).
    b. Testing part uses the learned model from the training part and evaluates how precise it is on the level of tiles (testing applies the trained model for a tile set and compares these results with the original annotations from raw dataset using the

[1] The masks are stored for optimization if the preprocessing phase is re-executed.
[2] In the new version of the pipeline, the gz-tables will not be stored as separate files, but will become part of the HDF5 index.

corresponding table inside the HDF5 index). Testing part uses a different subset of the input image data than the training part. Result of the testing part is a summary file containing a description about the model performance for each testing slide and a gz-table-new, containing the original columns (coordinates + label) and prediction result (a number between 0 and 1).

3. **Evaluation**. Goal of this part is evaluation of the trained model performance. While the testing part deals with evaluation on tile level, this part focuses on an evaluation of the whole scan. The evaluation part has two main inputs - 1) the raw dataset and 2) the result of the previous step (gz-table-new and the summary file). The gz-table-new and the summary file are used to generate a heatmap by iterating through the table and putting particular tiles together to build the whole scans (using the “prediction” column of the table, corresponding colour for the heatmap “tile” is determined). The “prediction result” column of the gzTableNew determines the colour of the resulting tile in the heatmap. In the consecutive step, the algorithm takes the heatmap (the gz-table-new would be enough, but slower than heatmap) and compares it with the annotations from the raw data set. Result of this step are statistics that are used for calculations of slide-level and tumour-level metrics, AUC and FROC score, respectively..

Provenance for the pipeline

Provenance information documenting execution of the pipeline is split into three bundles, each documenting a particular phase of the MLP (preprocessing, training & testing, evaluation). Machine learning specific semantics is attached to the provenance backbone.

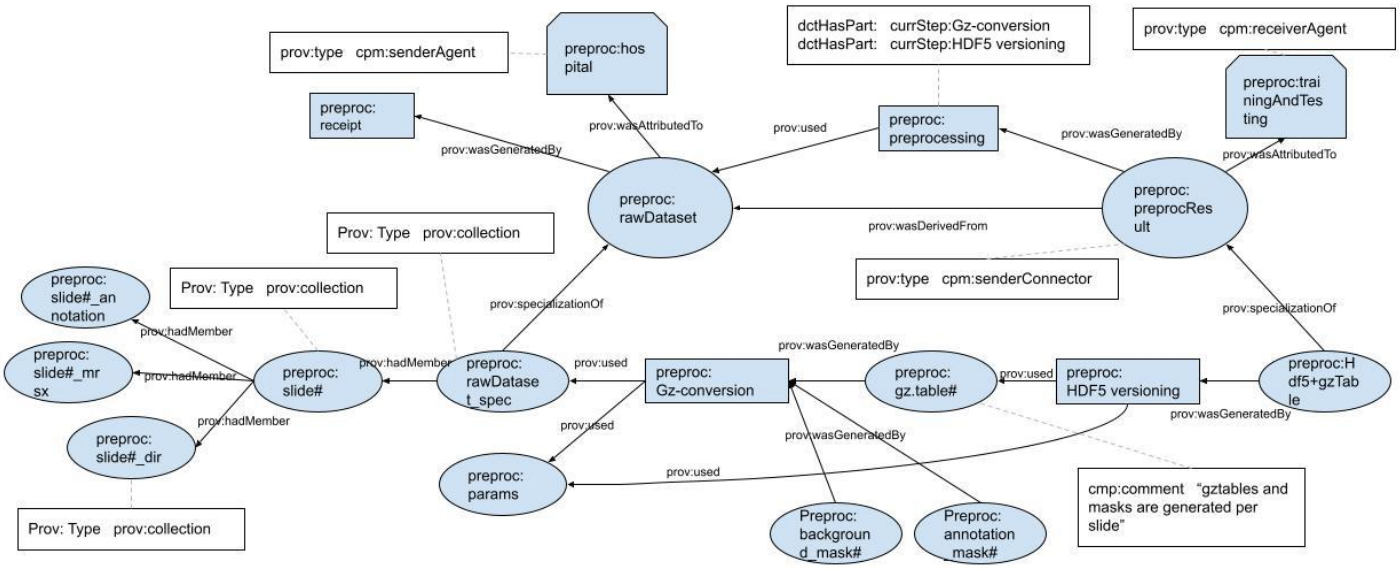
In the following paragraphs, we describe the most important provenance structures.

General notes:

- ‘#’ stands for a number in a provenance structure identifier. This means that particular structure can occur several times with different identifiers.
- All entities representing files or folders contain corresponding path and hash as attributes
- All entities/activities representing code contain gitCommitHash as an attribute
- Missing derivations where appropriate
- No machine learning ontology used yet
- Support of PIDs to be added
- Explicit expression of testing and training iterations is omitted in the simplified version
- Do we always want to replicate file system structure in PROV?
    - Activities do not necessarily use directories, they can use entities representing files
    - If all three phases are executed at once, we do not need to compute hashes in all three bundles
    - If all three phases are not executed at once, it would make sense to compute hashes of input files (especially raw_Dataset) individually for each bundle, since the files might change. Another thing is that if, for example, the testing part is run

individually, corresponding provenance should not be connected to the evaluation provenance, which could run weeks before.

Preprocessing bundle

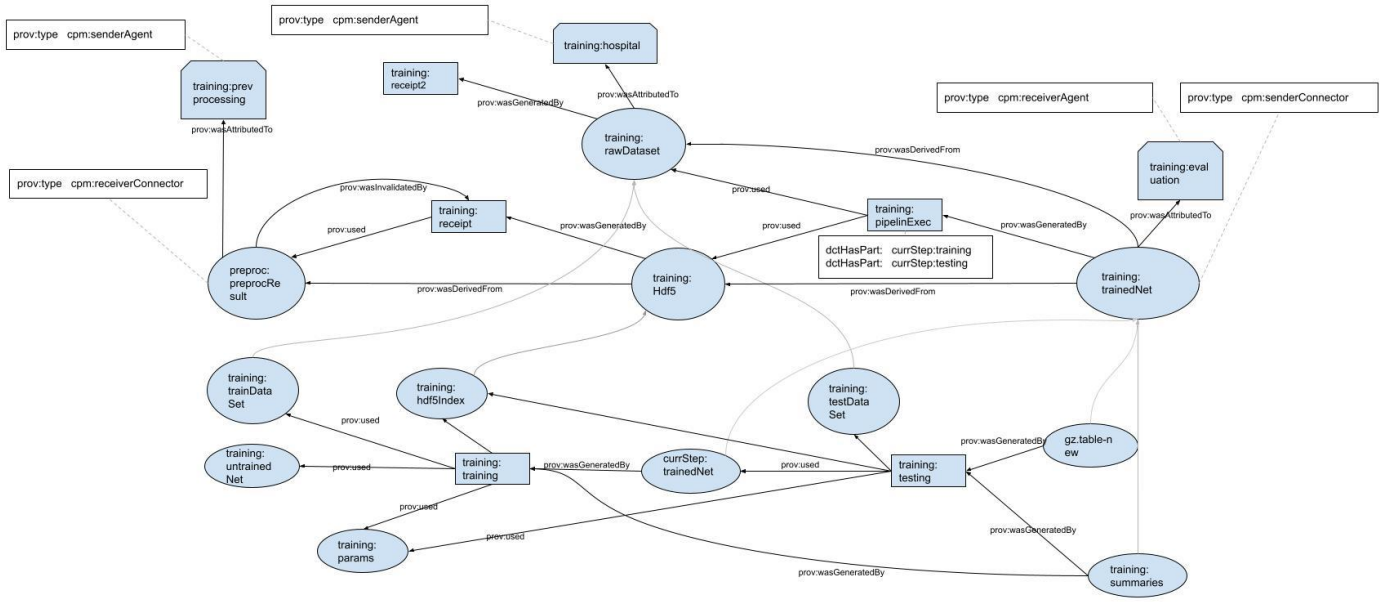


- preproc:rawDataset_Spec; Represents annotated scans from doctor as a collection of scans and annotations.
- preproc:gzConversion; An activity that is performed per slide individually. It represents computation of annotation masks, background masks and gz-table. The mapping between the masks and corresponding scan is created using file naming convention.
- preproc:hdf5Output - represents gz-tables which are indexed in HDF5

Provenance backbone notes

- Preprocessing bundle does not contain any entity of type cpm:senderConnector, since the previous step is not documented by provenance at the moment of writing this document. For that reason, the corresponding sender agent is attached to the entity which is derived from the receipt activity.
- Sender connector represents the output of this part - hdf5 file and gz-tables

Testing & training bundle



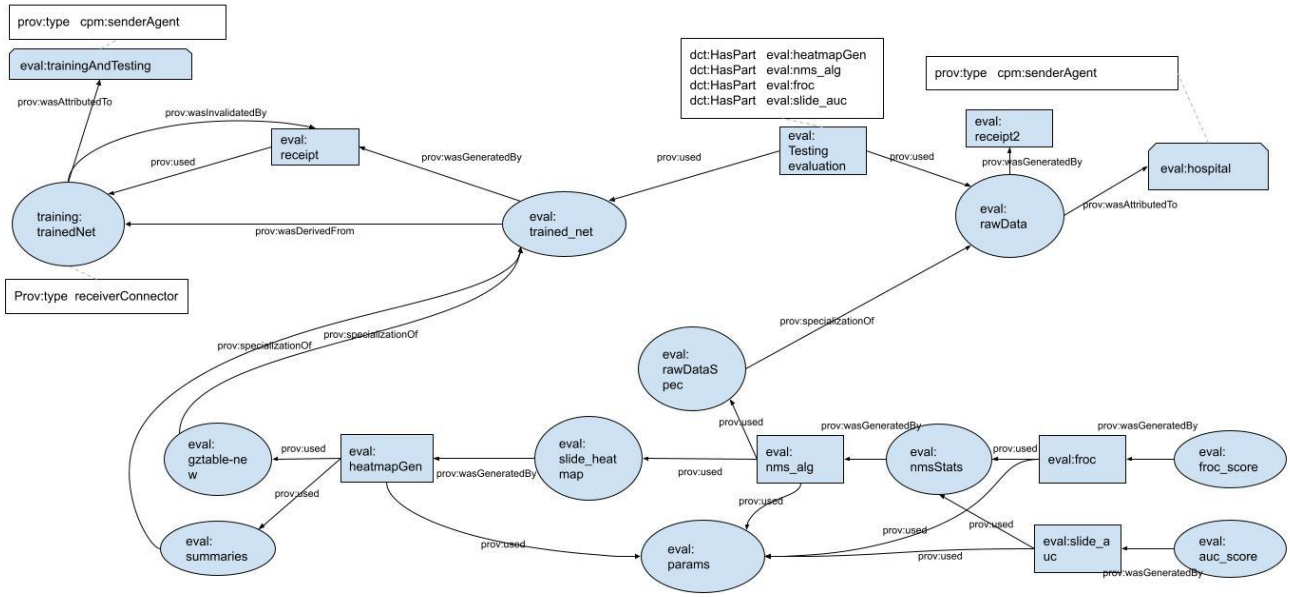
- training: training; represents the training process. The activity consumes four inputs (all represented as entities): parameters, training dataset (subset in the original dataset), the hdf5 file and untrained net. The untrained net is defined in a python file and config. The activity produces two outputs: trained net (which is represented as net weights file) and summaries. The activity may be expressed in provenance as a compound activity consisting of train epochs (omitted in the simplified version).
- training: train_epox_#; (omitted in the simplified version) documents details of each particular training epoch. Each training epoch consumes the ML model from the previous step and generated validation and training metrics. Another result of each epoch is a trained net with more precise estimations.
- training: trainDataset; represents image files that have been chosen for training part. In case that this provenance bundle should contain hashes of the input files, they may be expressed using the same way as in the “Preprocessing” bundle for the raw dataset.
- training: testing; represents the testing part; the testing part consumes four inputs (all represented as entities): parameters (the same parameter file as for training), test data set (subset of the raw dataset), the hdf5 file and the trained model (result of the training activity). The activity produces two outputs: test summaries and a gzTablenew. Testing is may be expressed in provenance as a compound activity consisting of activities documenting testing of particular slide (omitted in the simplified version).

Provenance backbone notes

- The receiver connector represents input coming from the previous (preprocessing) part of the pipeline, which is the hdf5 file and gz-tables
- The sender connector represents output of this part of the pipeline, which is a trained net, a summary file (containing details of training/testing) and an updated gzTable (called gzTableNew)

- The backbone contains one another input, which is not represented as a connector - it is the raw data set, coming from the hospital ().

Evaluation bundle



- eval:heatmapGen an activity representing the heatmap generation. Consumes two inputs: gzTableNew and testing summaries, generating the heatmap.
- eval:nmsAlg; an activity representing evaluation of the training result. It compares the heatmap with the raw dataset (by evaluating how the heatmap overlaps with the original annotations). Result of this activity is a statistics file
- eval:slAucact a eval:froc; activities that calculate qualitative evaluation of the training (using the statistics file generated by nmsAlg)

Provenance backbone notes

1. Input1: training:testResult - which is a result from the testing and training phase. Since the training & testing step is documented in a bundle, a particular receiver connector is present. A receiver agent is created for the connector.
2. Input2: eval:rawData -the raw dataset coming from hospital -> no corresponding receiver connector (as in the preprocessing or training part). Corresponding sender agent is thus related to an entity derived from the receipt activity (as in the preprocessing bundle).