In [1]:
```
import os
os.environ["OMP_NUM_THREADS"] = "1" #the number of threads to use for parallel regions
import flexynesis
import torch
torch.set_num_threads(4)
```

```
Seed set to 42
```

In [2]:
```
# parameters cell (required to pass arguments to the notebook) (see View -> show right sidebar -> add tag -> pa
HPO_ITER = 5 # number of HPO iterations for final modeling run
```

Modeling Breast Cancer Subtypes

Here, we demonstrate the capabilities of `flexynesis` on a multi-omic dataset of Breast Cancer samples from the METABRIC consortium. The data was downloaded from Cbioportal and randomly split into `train` (70% of the samples) and `test` (30% of the samples) data folders. The data files were processed to follow the same nomenclature.

- `gex.csv` contains "gene expression" data
- `cna.csv` contains "copy number alteration" data
- `mut.csv` contains "mutation" data, which is a binary matrix of genes versus samples.
- `clin.csv` contains "clinical/sample metatada", which is a table of clinical parameters such as age, gender, therapy, subtypes.

Data Download

The data can be downloaded as follows:

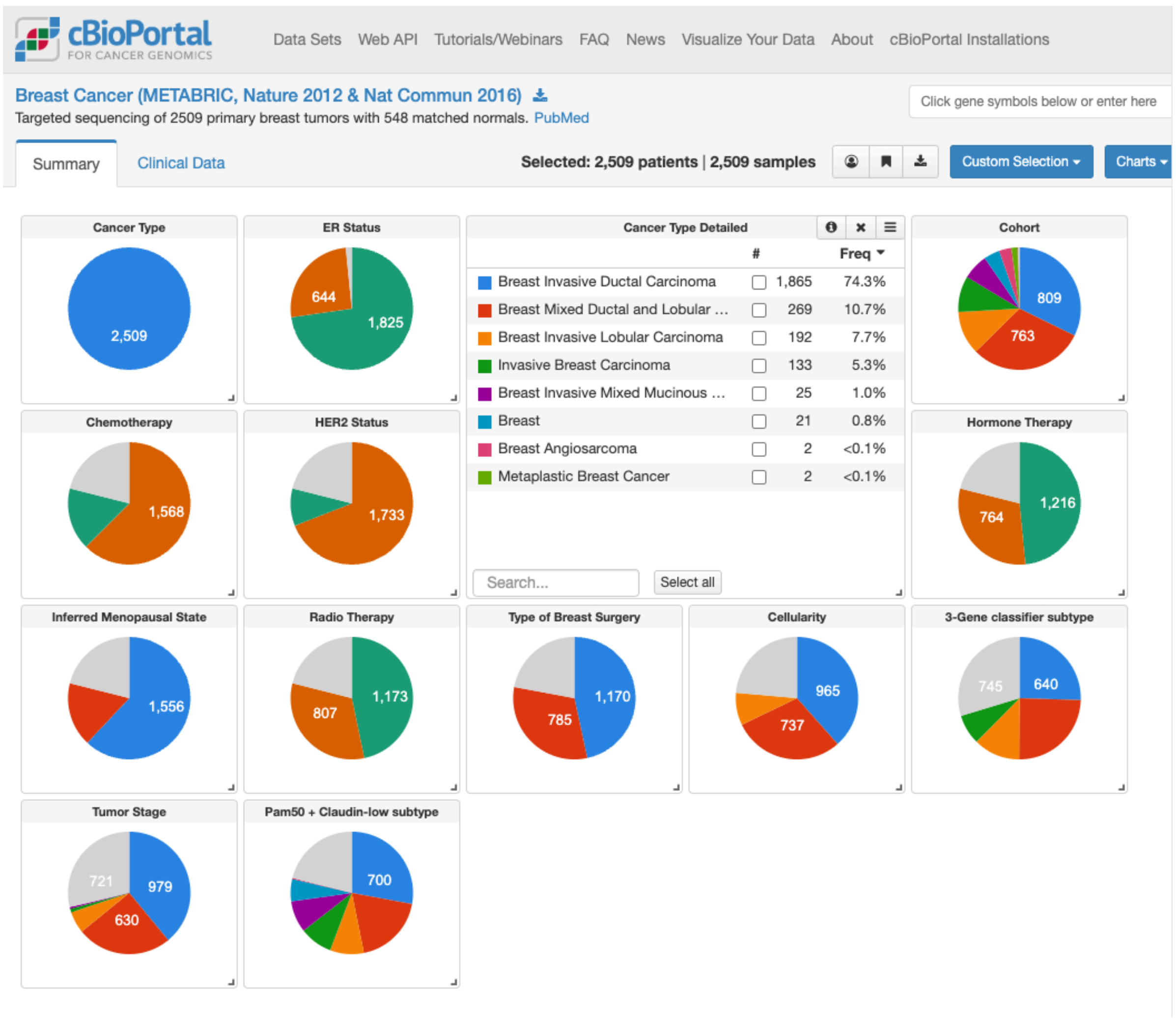
In [3]:
```
if not os.path.exists("brca_metabric_processed"):
    !wget -O brca_metabric.tgz "https://bimsbstatic.mdc-berlin.de/akalin/buyar/flexynesis-benchmark-datasets/br
```

```
--2025-03-10 22:33:02--  https://bimsbstatic.mdc-berlin.de/akalin/buyar/flexynesis-benchmark-datasets/brca_metab
ric_processed.tgz
Resolving bimsbstatic.mdc-berlin.de (bimsbstatic.mdc-berlin.de)... 141.80.181.46, 141.80.181.47
connected. to bimsbstatic.mdc-berlin.de (bimsbstatic.mdc-berlin.de)|141.80.181.46|:443...
HTTP request sent, awaiting response... 200 OK
Length: 407225158 (388M) [application/octet-stream]
Saving to: ‘brca_metabric.tgz’

brca_metabric.tgz   100%[===================>] 388.36M   207MB/s    in 1.9s

2025-03-10 22:33:04 (207 MB/s) - ‘brca_metabric.tgz’ saved [407225158/407225158]

brca_metabric_processed/
brca_metabric_processed/test/
brca_metabric_processed/test/gex.csv
brca_metabric_processed/test/mut.csv
brca_metabric_processed/test/clin.csv
brca_metabric_processed/test/cna.csv
brca_metabric_processed/9606.protein.aliases.v12.0.txt.gz
brca_metabric_processed/9606.protein.links.v12.0.txt.gz
brca_metabric_processed/train/
brca_metabric_processed/train/gex.csv
brca_metabric_processed/train/mut.csv
brca_metabric_processed/train/clin.csv
brca_metabric_processed/train/cna.csv
```



Breast Cancer (METABRIC, Nature 2012 & Nat Commun 2016)
Targeted sequencing of 2509 primary breast tumors with 548 matched normals. PubMed

Selected: 2,509 patients | 2,509 samples

Cancer Type Detailed

		#	Freq
🟦	Breast Invasive Ductal Carcinoma	1,865	74.3%
🟥	Breast Mixed Ductal and Lobular ...	269	10.7%
🟧	Breast Invasive Lobular Carcinoma	192	7.7%
🟩	Invasive Breast Carcinoma	133	5.3%
🟪	Breast Invasive Mixed Mucinous ...	25	1.0%
🟦	Breast	21	0.8%
🟥	Breast Angiosarcoma	2	<0.1%
🟩	Metaplastic Breast Cancer	2	<0.1%

Let's check the number of samples and number of features in the corresponding files under train and test folders:

```
In [4]: # train data
        !wc -l ./brca_metabric_processed/train/*
        # wc: word count: count the number of lines
```

```
   1307 ./brca_metabric_processed/train/clin.csv
  22543 ./brca_metabric_processed/train/cna.csv
  20604 ./brca_metabric_processed/train/gex.csv
    174 ./brca_metabric_processed/train/mut.csv
  44628 total
```

```
In [5]: # test data
        !wc -l ./brca_metabric_processed/test/*
```

```
    561 ./brca_metabric_processed/test/clin.csv
  22543 ./brca_metabric_processed/test/cna.csv
  20604 ./brca_metabric_processed/test/gex.csv
    174 ./brca_metabric_processed/test/mut.csv
  43882 total
```

Importing Multiomics Data Into Flexynesis

Procedure

We use the `flexynesis.DataImporter` class to import multiomics data from the data folders. Data importing includes:

1. Validation of the data folders
2. Reading data matrices
3. Data processing, which includes:
    - Cleaning up the data matrices to:
        - remove uninformative features (e.g. features with near-zero-variation)
        - remove samples with too many NA values
        - remove features with too many NA values and impute NA values for the rest with few NA values
4. Feature selection **only on training data** for each omics layer separately:
    - Features are sorted by Laplacian score

- Features that make it in the `top_percentile`
- Highly redundant features are further removed (for a pair of highly correlated features, keep the one with the higher laplacian score).

5. Harmonize the training data with the test data.
    - Subset the test data features to those that are kept for training data
6. Normalize the datasets
    - Normalize training data (standard scaling) and apply the same scaling factors to the test data.
7. (Optional): Log transform the final matrices.
8. Distinguish numerical and categorical variables in the "clin.csv" file. For categorical variables, create a numerical encoding of the labels for training data. Use the same encoders to map the test samples to the same numerical encodings.

Usage

- Here, we import both train/test datasets from the data folder we downloaded and unpacked before.
- We choose which omic layers to import
- We choose whether we want to concatenate the data matrices (early integration) or not (intermediate integration) before running them through the neural networks.
- We want to apply feature selection and keep only top 10% of the features. In the end, we want to keep at least 1000 features per omics layer.
- We apply a variance threshold (for simplicity of demonstration, we want to keep a small number of most variable features). Setting this to 80, will remove 80% of the features with lowest variation from each modality.

In [6]:
```
data_importer = flexynesis.DataImporter(path ='./brca_metabric_processed/',
                                        data_types = ['gex', 'cna'],
                                        concatenate=False,
                                        top_percentile=10,
                                        min_features=1000,
                                        variance_threshold=0.8, # set to 0.8 for 80%
                                       )
train_dataset, test_dataset = data_importer.import_data()
```

```
[INFO] ================= Importing Data =================
[INFO] Validating data folders...

[INFO] ----------------- Reading Data -----------------
[INFO] Importing ./brca_metabric_processed/train/clin.csv...
[INFO] Importing ./brca_metabric_processed/train/gex.csv...
[INFO] Importing ./brca_metabric_processed/train/cna.csv...

[INFO] ----------------- Reading Data -----------------
[INFO] Importing ./brca_metabric_processed/test/clin.csv...
[INFO] Importing ./brca_metabric_processed/test/gex.csv...
[INFO] Importing ./brca_metabric_processed/test/cna.csv...

[INFO] ----------------- Checking for problems with the input data -----------------
[INFO] Data structure is valid with no errors or warnings.

[INFO] ----------------- Processing Data (train) -----------------

[INFO] ----------------- Cleaning Up Data -----------------

[INFO] working on layer:  gex
[INFO] Imputing NA values to median of features, affected # of cells in the matrix 7  # of rows: 5
[INFO] Number of NA values:  0
[INFO] DataFrame gex - Removed 16482 features.

[INFO] working on layer:  cna
[INFO] Imputing NA values to median of features, affected # of cells in the matrix 108  # of rows: 87
[INFO] Number of NA values:  0
[INFO] DataFrame cna - Removed 18033 features.
[INFO] DataFrame gex - Removed 3 samples (0.23%).
[INFO] DataFrame cna - Removed 3 samples (0.23%).
[INFO] Implementing feature selection using laplacian score for layer: gex with  4121 features  and  1303  sampl
es
Calculating Laplacian scores: 100%|██████████| 4121/4121 [00:01<00:00, 2248.90it/s]
Filtering redundant features: 100%|██████████| 1000/1000 [00:00<00:00, 4197.83it/s]
[INFO] Implementing feature selection using laplacian score for layer: cna with  4509 features  and  1303  sampl
es
Calculating Laplacian scores: 100%|██████████| 4509/4509 [00:02<00:00, 2248.80it/s]
Filtering redundant features: 100%|██████████| 1000/1000 [00:00<00:00, 241788.44it/s]
```

```
[INFO] ----------------- Processing Data (test) -----------------

[INFO] ----------------- Cleaning Up Data -----------------

[INFO] working on layer:  gex
[INFO] Number of NA values:  0
[INFO] DataFrame gex - Removed 16482 features.

[INFO] working on layer:  cna
[INFO] Imputing NA values to median of features, affected # of cells in the matrix 63  # of rows: 51
[INFO] Number of NA values:  0
[INFO] DataFrame cna - Removed 18033 features.
[INFO] DataFrame gex - Removed 2 samples (0.36%).
[INFO] DataFrame cna - Removed 2 samples (0.36%).

[INFO] ----------------- Harmonizing Data Sets -----------------

[INFO] ----------------- Finished Harmonizing -----------------

[INFO] ----------------- Normalizing Data -----------------

[INFO] ----------------- Normalizing Data -----------------
[INFO] Training Data Stats:  {'feature_count in: cna': 1000, 'feature_count in: gex': 977, 'sample_count': 1303}
[INFO] Test Data Stats:  {'feature_count in: cna': 1000, 'feature_count in: gex': 977, 'sample_count': 558}
[INFO] Merging Feature Logs...
[INFO] Data import successful.
```

- **dataset.dat** contains the data matrices

In [7]:
```
train_dataset.dat
```

Out[7]:
```
{'cna': tensor([[ 0.2908,  0.2509,  0.2583,  ...,  0.4128,  0.4383,  0.4398],
         [-1.0024, -1.0362, -1.0212,  ..., -0.8992,  1.6087,  1.6021],
         [ 0.2908,  0.2509,  0.2583,  ...,  0.4128, -0.7321, -0.7226],
         ...,
         [ 1.5840,  1.5381,  1.5378,  ...,  1.7248, -0.7321, -0.7226],
         [ 1.5840,  1.5381,  1.5378,  ...,  1.7248, -0.7321, -0.7226],
         [-1.0024, -1.0362, -1.0212,  ..., -0.8992,  1.6087,  1.6021]]),
 'gex': tensor([[ 0.5153,  0.1786, -0.0848,  ..., -0.9461,  1.1264,  1.1611],
         [ 0.3813,  0.5253,  0.8374,  ..., -0.5427, -0.4600, -0.6756],
         [-2.9466, -2.5524, -1.8277,  ...,  2.6105,  0.0816,  0.6697],
         ...,
         [ 0.7351,  0.7477,  0.0686,  ..., -0.4098, -0.0152,  0.1380],
         [ 0.6964,  1.0380,  0.4544,  ..., -0.6631,  0.6417,  1.3360],
         [ 0.8138,  0.8124,  0.8440,  ..., -0.8887,  1.4068,  1.4723]])}
```

In [8]:
```
train_dataset.dat['gex'].shape, train_dataset.dat['cna'].shape
```

Out[8]:
```
(torch.Size([1303, 977]), torch.Size([1303, 1000]))
```

- dataset.ann contains the sample annotation data (from clin.csv), where the keys are variable names and values are tensors.

In [9]:
```
train_dataset.ann
```

Out[9]:
```
{'LYMPH_NODES_EXAMINED_POSITIVE': tensor([ 0,  0, 11,  ...,  3,  0,  0]),
 'NPI': tensor([4.0540, 3.0360, 6.0020,  ..., 4.1140, 3.0400, 1.0500],
        dtype=torch.float64),
 'AGE_AT_DIAGNOSIS': tensor([58.2600, 61.9200, 64.6800,  ..., 72.8800, 66.0100, 59.2200],
        dtype=torch.float64),
 'OS_MONTHS': tensor([ 49.4667, 186.4000,  83.6667,  ...,  28.8333,  26.3333, 211.1333],
        dtype=torch.float64),
 'RFS_MONTHS': tensor([ 33.5200,  79.2800,  82.5700,  ...,  13.7500,  17.7000, 208.3600],
        dtype=torch.float64),
 'CELLULARITY': tensor([2., 2., 0.,  ..., 2., 1., 0.], dtype=torch.float64),
 'CHEMOTHERAPY': tensor([1., 0., 1.,  ..., 0., 0., 0.], dtype=torch.float64),
 'COHORT': tensor([3., 2., 2.,  ..., 2., 2., 4.], dtype=torch.float64),
 'ER_IHC': tensor([1., 1., 0.,  ..., 1., 1., 1.], dtype=torch.float64),
 'HER2_SNP6': tensor([2., 2., 2.,  ..., 2., 2., 0.], dtype=torch.float64),
 'HORMONE_THERAPY': tensor([1., 0., 1.,  ..., 1., 0., 0.], dtype=torch.float64),
 'INFERRED_MENOPAUSAL_STATE': tensor([0., 0., 0.,  ..., 0., 0., 0.], dtype=torch.float64),
 'SEX': tensor([0., 0., 0.,  ..., 0., 0., 0.], dtype=torch.float64),
 'INTCLUST': tensor([10., 10.,  1.,  ...,  3.,  9.,  0.], dtype=torch.float64),
 'OS_STATUS': tensor([1., 1., 1.,  ..., 1., 1., 1.], dtype=torch.float64),
 'CLAUDIN_SUBTYPE': tensor([1., 2., 0.,  ..., 2., 3., 3.], dtype=torch.float64),
 'THREEGENE': tensor([0., 0., 2.,  ..., 1., 0., 0.], dtype=torch.float64),
 'VITAL_STATUS': tensor([0., 1., 1.,  ..., 0., 1., 0.], dtype=torch.float64),
 'LATERALITY': tensor([nan, 0., 1.,  ..., 1., 0., 0.], dtype=torch.float64),
 'RADIO_THERAPY': tensor([0., 0., 1.,  ..., 1., 1., 0.], dtype=torch.float64),
 'HISTOLOGICAL_SUBTYPE': tensor([0., 0., 2.,  ..., 4., 1., 0.], dtype=torch.float64),
 'BREAST_SURGERY': tensor([1., 1., 1.,  ..., 1., 0., 1.], dtype=torch.float64),
 'RFS_STATUS': tensor([1., 1., 0.,  ..., 1., 1., 1.], dtype=torch.float64)}
```

- A mapping of the sample labels for categorical variables can be found in dataset.label_mappings

In [10]:
```
train_dataset.label_mappings
```

Out[10]:
```
{'CELLULARITY': {0: 'High', 1: 'Low', 2: 'Moderate', 3: nan},
 'CHEMOTHERAPY': {0: 'NO', 1: 'YES'},
 'COHORT': {0: 'cohort1',
  1: 'cohort2',
  2: 'cohort3',
  3: 'cohort4',
  4: 'cohort5'},
 'ER_IHC': {0: 'Negative', 1: 'Positve', 2: nan},
 'HER2_SNP6': {0: 'GAIN', 1: 'LOSS', 2: 'NEUTRAL', 3: 'UNDEF'},
 'HORMONE_THERAPY': {0: 'NO', 1: 'YES'},
 'INFERRED_MENOPAUSAL_STATE': {0: 'Post', 1: 'Pre'},
 'SEX': {0: 'Female'},
 'INTCLUST': {0: '1',
  1: '10',
  2: '2',
  3: '3',
  4: '4ER+',
  5: '4ER-',
  6: '5',
  7: '6',
  8: '7',
  9: '8',
  10: '9'},
 'OS_STATUS': {0: '0:LIVING', 1: '1:DECEASED'},
 'CLAUDIN_SUBTYPE': {0: 'Basal',
  1: 'Her2',
  2: 'LumA',
  3: 'LumB',
  4: 'NC',
  5: 'Normal',
  6: 'claudin-low'},
 'THREEGENE': {0: 'ER+/HER2- High Prolif',
  1: 'ER+/HER2- Low Prolif',
  2: 'ER-/HER2-',
  3: 'HER2+',
  4: nan},
 'VITAL_STATUS': {0: 'Died of Disease',
  1: 'Died of Other Causes',
  2: 'Living',
  3: nan},
 'LATERALITY': {0: 'Left', 1: 'Right', 2: nan},
 'RADIO_THERAPY': {0: 'NO', 1: 'YES'},
 'HISTOLOGICAL_SUBTYPE': {0: 'Ductal/NST',
  1: 'Lobular',
  2: 'Medullary',
  3: 'Metaplastic',
  4: 'Mixed',
  5: 'Mucinous',
  6: 'Other',
  7: 'Tubular/ cribriform',
  8: nan},
 'BREAST_SURGERY': {0: 'BREAST CONSERVING', 1: 'MASTECTOMY', 2: nan},
 'RFS_STATUS': {0: '0:Not Recurred', 1: '1:Recurred', 2: nan}}
```

- As the data matrices are stored as tensors, the row and column names cannot be stored as tensors. These are stored in the same dataset object as: `dataset.samples` and `dataset.features`

In [11]:
```
train_dataset.samples[1:10], train_dataset.features
```

Out[11]:
```
(['MB-5098',
  'MB-4714',
  'MB-7063',
  'MB-4941',
  'MB-4004',
  'MB-5211',
  'MB-6200',
  'MB-4694',
  'MB-5395'],
 {'cna': Index(['DAP3', 'FCRLA', 'TOP1P1', 'LAMC1', 'TDRKH', 'LBR', 'CRB1', 'LINC00467',
         'DSTYK', 'FMO5',
         ...
         'C16orf90', 'ADSS', 'ABCA17P', 'DESI2', 'IL19', 'ZNF16', 'TONSL',
         'HFE2', 'DENND3', 'HSF1'],
        dtype='object', length=1000),
  'gex': Index(['FOXA1', 'MLPH', 'ESR1', 'GATA3', 'SPDEF', 'TBC1D9', 'FOXC1', 'C1S',
         'XBP1', 'CA12',
         ...
         'EXO5', 'CHRNA5', 'TRIM13', 'SHROOM4', 'CYB5RL', 'CDKN2AIPNL', 'TMEM17',
         'CD19', 'TDP1', 'ZNF549'],
        dtype='object', length=977)})
```

- We can get a summary of sample metadata using `print_summary_stats`. For categorical variables, we can the sample counts per label and for numerical variables, we get mean/median statistics.

In [12]:
```
flexynesis.print_summary_stats(train_dataset)
```

```
Summary for variable: LYMPH_NODES_EXAMINED_POSITIVE
Numerical Variable Summary: Median = 0.0, Mean = 1.9286262471220261
------
Summary for variable: NPI
Numerical Variable Summary: Median = 4.04, Mean = 4.017291158864159
------
Summary for variable: AGE_AT_DIAGNOSIS
Numerical Variable Summary: Median = 61.79, Mean = 61.30643898695319
------
Summary for variable: OS_MONTHS
Numerical Variable Summary: Median = 114.4666667, Mean = 125.03573804066693
------
Summary for variable: RFS_MONTHS
Numerical Variable Summary: Median = 100.63, Mean = 109.94034535686878
------
Summary for variable: CELLULARITY
Categorical Variable Summary:
  Label: High, Count: 656
  Label: Low, Count: 136
  Label: Moderate, Count: 484
  Label: nan, Count: 27
------
Summary for variable: CHEMOTHERAPY
Categorical Variable Summary:
  Label: NO, Count: 1044
  Label: YES, Count: 259
------
Summary for variable: COHORT
Categorical Variable Summary:
  Label: cohort1, Count: 308
  Label: cohort2, Count: 196
  Label: cohort3, Count: 521
  Label: cohort4, Count: 159
  Label: cohort5, Count: 119
------
Summary for variable: ER_IHC
Categorical Variable Summary:
  Label: Negative, Count: 289
  Label: Positve, Count: 994
  Label: nan, Count: 20
------
Summary for variable: HER2_SNP6
Categorical Variable Summary:
  Label: GAIN, Count: 279
  Label: LOSS, Count: 67
  Label: NEUTRAL, Count: 955
  Label: UNDEF, Count: 2
------
Summary for variable: HORMONE_THERAPY
Categorical Variable Summary:
  Label: NO, Count: 508
  Label: YES, Count: 795
------
Summary for variable: INFERRED_MENOPAUSAL_STATE
Categorical Variable Summary:
  Label: Post, Count: 1034
  Label: Pre, Count: 269
------
Summary for variable: SEX
Categorical Variable Summary:
  Label: Female, Count: 1303
------
Summary for variable: INTCLUST
Categorical Variable Summary:
  Label: 1, Count: 93
  Label: 10, Count: 151
  Label: 2, Count: 50
  Label: 3, Count: 195
  Label: 4ER+, Count: 154
  Label: 4ER-, Count: 50
  Label: 5, Count: 127
  Label: 6, Count: 60
  Label: 7, Count: 129
  Label: 8, Count: 197
  Label: 9, Count: 97
------
Summary for variable: OS_STATUS
Categorical Variable Summary:
  Label: 0:LIVING, Count: 539
  Label: 1:DECEASED, Count: 764
------
Summary for variable: CLAUDIN_SUBTYPE
Categorical Variable Summary:
  Label: Basal, Count: 145
  Label: Her2, Count: 152
  Label: LumA, Count: 468
  Label: LumB, Count: 328
  Label: NC, Count: 5
  Label: Normal, Count: 90
  Label: claudin-low, Count: 115
------
Summary for variable: THREEGENE
Categorical Variable Summary:
  Label: ER+/HER2- High Prolif, Count: 414
  Label: ER+/HER2- Low Prolif, Count: 425
  Label: ER-/HER2-, Count: 197
  Label: HER2+, Count: 128
  Label: nan, Count: 139
------
Summary for variable: VITAL_STATUS
Categorical Variable Summary:
  Label: Died of Disease, Count: 432
  Label: Died of Other Causes, Count: 331
  Label: Living, Count: 539
  Label: nan, Count: 1
------
Summary for variable: LATERALITY
Categorical Variable Summary:
  Label: Left, Count: 649
  Label: Right, Count: 588
  Label: nan, Count: 66
------
Summary for variable: RADIO_THERAPY
Categorical Variable Summary:
  Label: NO, Count: 522
  Label: YES, Count: 781
------
Summary for variable: HISTOLOGICAL_SUBTYPE
Categorical Variable Summary:
  Label: Ductal/NST, Count: 999
  Label: Lobular, Count: 98
  Label: Medullary, Count: 18
  Label: Metaplastic, Count: 1
  Label: Mixed, Count: 135
  Label: Mucinous, Count: 15
  Label: Other, Count: 12
  Label: Tubular/ cribriform, Count: 15
  Label: nan, Count: 10
------
Summary for variable: BREAST_SURGERY
Categorical Variable Summary:
  Label: BREAST CONSERVING, Count: 524
  Label: MASTECTOMY, Count: 763
  Label: nan, Count: 16
------
Summary for variable: RFS_STATUS
Categorical Variable Summary:
  Label: 0:Not Recurred, Count: 763
  Label: 1:Recurred, Count: 539
  Label: nan, Count: 1
------
```

Training flexynesis models

We create a `tuner` object by specifying:

1. `dataset` : the training dataset (as we constructed above)
2. `model_class` : which model architecture to use: a) DirectPred: a fully connected network (standard multilayer perceptron) with supervisor heads (one MLP for each target variable) b) Supervised Variational Autoencoder: A variational autoencoder (MMD-loss) with supervisor heads (one MLP for each target variable) c) MultiTripletNetwork: A network structured in triplets to enable contrastive learning (using triplet loss) and additiona supervisor heads (one MLP for each target variable)
3. `target_variables` : A comma separated list of target variables (specify the column headers from the clin.csv).
   - One MLP per each target variable will be created.
   - The target variables may contain NA values
4. `config_name` : which hyperparameter search space configuration to use.
5. `n_iter` : How many hyperparameter search steps to implement.

- This example runs 1 hyperparameter search step using DirectPred architecture and a hyperparameter configuration space defined for "DirectPred" with a supervisor head for "CLAUDIN_SUBTYPE" variable:

tuner = flexynesis.HyperparameterTuning(dataset = train_dataset, model_class = flexynesis.DirectPred, target_variables = ["CLAUDIN_SUBTYPE"], config_name = "DirectPred", n_iter=1)

- We use `perform_tuning` function to run the hyperparameter optimisation procedure. At the end of the parameter optimisation, best model will be selected and returned.

model, best_params = tuner.perform_tuning()

Early Stopping

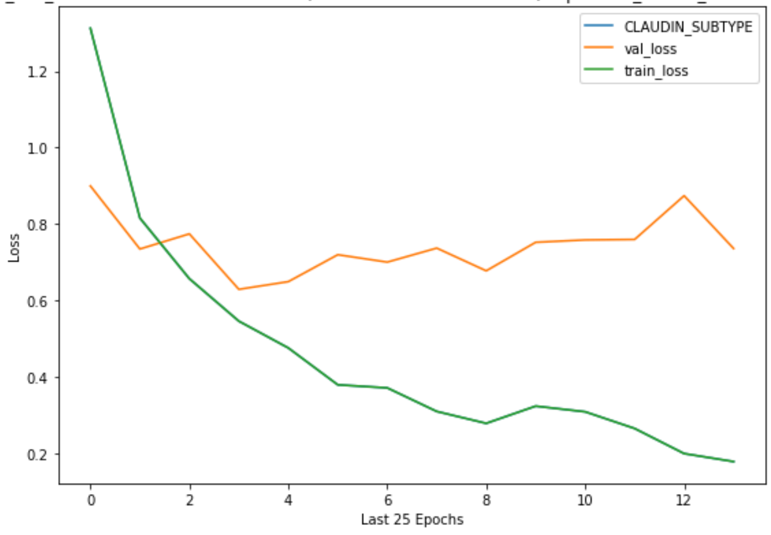
Training a model longer than needed causes the model to overfit, yield worse validation performance, and also it takes a longer time to train the models, considering if we have to run a long hyperparameter optimisation routine, not just for 1 step, but say more than 100 steps.

It is possible to set `early stopping` criteria in flexynesis, which is basically a simple callback that is handled by `Pytorch Lightning`. This is regulated using the `early_stop_patience`. When set to e.g. 10, the training will stop if the validation loss has not been improved in the last 10 epochs.

One can also visualize the training setting `plot_losses` to `True`. This will print the loss values training/validation splits and also the individual loss values for each target variable. In this case, the total loss value for the training equals the loss value of the single variable we chose.

In [13]:
```python
tuner = flexynesis.HyperparameterTuning(dataset = train_dataset,
                                        model_class = flexynesis.DirectPred,
                                        target_variables = ["CLAUDIN_SUBTYPE"],
                                        config_name = "DirectPred",
                                        n_iter=1,  plot_losses=True, early_stop_patience=10)
model, best_params = tuner.perform_tuning()
```

HPO Step=1 out of 1
(latent_dim=105, hidden_dim_factor=0.25503043695984917, lr=0.00362561763457623, supervisor_hidden_dim=22, epochs=500, batch_size=32)



```
Validation: |          | 0/? [00:00<?, ?it/s]
```

Validate metric	DataLoader 0
CLAUDIN_SUBTYPE	0.7358713746070862
val_loss	0.7358713746070862

```
Tuning Progress: 100%|██████████| 1/1 [00:12<00:00, 12.05s/it, Iteration=1, Best Loss=0.736]
[INFO] current best val loss: 0.7358713746070862; best params: {'latent_dim': 105, 'hidden_dim_factor': 0.255030
43695984917, 'lr': 0.00362561763457623, 'supervisor_hidden_dim': 22, 'epochs': 500, 'batch_size': 32} since 0 hp
o iterations
```

- One can also provide own parameter optimisation spaces via a `yaml` file as input:

tuner = flexynesis.HyperparameterTuning(dataset = train_dataset, model_class = flexynesis.DirectPred, target_variables = ["CLAUDIN_SUBTYPE"], config_name = "DirectPred", n_iter=1, plot_losses=True, config_path='./conf.yaml') model, best_params = tuner.perform_tuning()

- We can also provide multiple target variables as input. This will create multiple MLP heads (one per variable) and the network will be trained to learn to predict both variables.

tuner = flexynesis.HyperparameterTuning(dataset = train_dataset, model_class = flexynesis.DirectPred, target_variables = ["CLAUDIN_SUBTYPE, "CHEMOTHERAPY"], config_name = "DirectPred", n_iter=1, plot_losses=True, early_stop_patience=10) model, best_params = tuner.perform_tuning()

- We can mix numerical and categorical variables. The relevant network structure and evaluation procedures will be applied depending on the type of variable

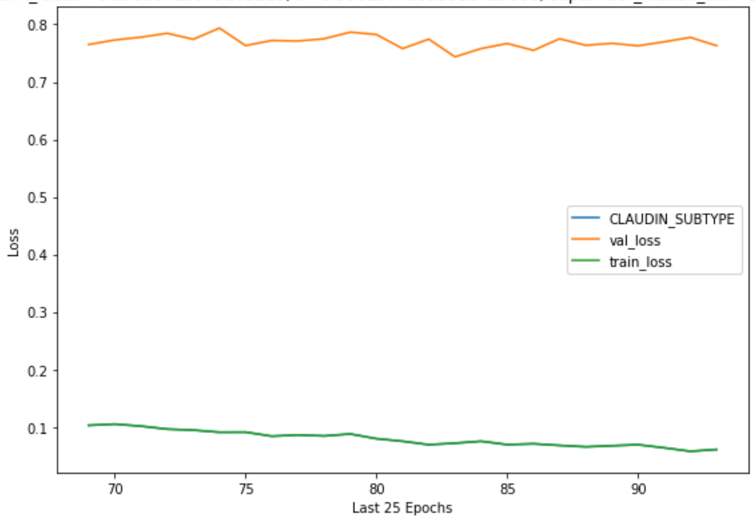
tuner = flexynesis.HyperparameterTuning(dataset = train_dataset, model_class = flexynesis.DirectPred, target_variables = ["CLAUDIN_SUBTYPE, "CHEMOTHERAPY", "LYMPH_NODES_EXAMINED_POSITIVE"], config_name = "DirectPred", n_iter=1, plot_losses=True, early_stop_patience=10) model, best_params = tuner.perform_tuning()

Longer Training

In reality, hyperparameter optimisation should run for multiple steps so that the parameter search space is large enough to find a good set. However, for demonstration purposes, we only run it for 5 steps here.

In [14]:
```
tuner = flexynesis.HyperparameterTuning(dataset = train_dataset,
                                        model_class = flexynesis.DirectPred,
                                        target_variables = ["CLAUDIN_SUBTYPE"],
                                        config_name = "DirectPred",
                                        n_iter=HPO_ITER, plot_losses=True,
                                        early_stop_patience=10)
model, best_params = tuner.perform_tuning()
```

HPO Step=5 out of 5
(latent_dim=107, hidden_dim_factor=0.29138413075201125, lr=0.00015679933916723006, supervisor_hidden_dim=24, epochs=500, batch_size=64)



```
Validation: |          | 0/? [00:00<?, ?it/s]
```

Validate metric	DataLoader 0
CLAUDIN_SUBTYPE val_loss	0.763045072555542 0.763045072555542

```
Tuning Progress: 100%|██████████| 5/5 [02:13<00:00, 26.60s/it, Iteration=5, Best Loss=0.648]
[INFO] current best val loss: 0.6481812596321106; best params: {'latent_dim': 82, 'hidden_dim_factor': 0.2139351
2381599933, 'lr': 0.001640928673064793, 'supervisor_hidden_dim': 12, 'epochs': 500, 'batch_size': 128} since 1 h
po iterations
```

In [15]:
```
model
```

Out[15]:
```
DirectPred(
  (log_vars): ParameterDict(  (CLAUDIN_SUBTYPE): Parameter containing: [torch.FloatTensor of size 1])
  (encoders): ModuleList(
    (0): MLP(
      (layer_1): Linear(in_features=1000, out_features=213, bias=True)
      (layer_out): Linear(in_features=213, out_features=82, bias=True)
      (relu): ReLU()
      (dropout): Dropout(p=0.1, inplace=False)
      (batchnorm): BatchNorm1d(213, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): MLP(
      (layer_1): Linear(in_features=977, out_features=209, bias=True)
      (layer_out): Linear(in_features=209, out_features=82, bias=True)
      (relu): ReLU()
      (dropout): Dropout(p=0.1, inplace=False)
      (batchnorm): BatchNorm1d(209, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (MLPs): ModuleDict(
    (CLAUDIN_SUBTYPE): MLP(
      (layer_1): Linear(in_features=164, out_features=12, bias=True)
      (layer_out): Linear(in_features=12, out_features=7, bias=True)
      (relu): ReLU()
      (dropout): Dropout(p=0.1, inplace=False)
      (batchnorm): BatchNorm1d(12, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
)
```

In [16]:
```
best_params
```

```
Out[16]: {'latent_dim': 82,
          'hidden_dim_factor': 0.21393512381599933,
          'lr': 0.001640928673064793,
          'supervisor_hidden_dim': 12,
          'epochs': 49,
          'batch_size': 128}
```

Prediction and Model Evaluation

We can use the best model (chosen based on the hyperparameter optimisation procedure) to make predictions on the test dataset

```
In [17]: y_pred_dict = model.predict(test_dataset)
```

```
In [18]: y_pred_dict
```

```
Out[18]: {'CLAUDIN_SUBTYPE': array([[2.0163823e-03, 6.8619982e-03, 9.5653510e-01, ..., 2.8585349e-03,
                  3.3545003e-03, 6.5585701e-03],
                 [9.7966527e-05, 6.1605322e-05, 4.0692952e-04, ..., 1.7122808e-04,
                  7.6720040e-05, 1.6082537e-04],
                 [2.2929674e-02, 5.7486701e-01, 9.2111528e-02, ..., 1.1690882e-02,
                  2.4324028e-01, 1.3085915e-02],
                 ...,
                 [1.0378690e-04, 7.8637015e-05, 4.0691765e-04, ..., 2.2710842e-04,
                  9.2869806e-05, 2.1998209e-04],
                 [4.2058967e-02, 2.0338779e-02, 1.5233769e-02, ..., 1.0042601e-02,
                  2.8719392e-03, 8.8782912e-01],
                 [2.8028842e-02, 2.6186591e-01, 5.2563328e-02, ..., 2.2693027e-02,
                  2.3881942e-02, 4.3592494e-02]], dtype=float32)}
```

- The predictions are class labels for both variables. Now, we can run `evaluate_wrapper` to evaluate all predictions. The wrapper goes through each variable and figures out which type of evaluation to apply to the corresponding variable (whether to report metrics relevant to regression tasks or classification tasks)

```
In [19]: metrics_df = flexynesis.evaluate_wrapper(method = 'DirectPred', y_pred_dict=y_pred_dict, dataset = test_dataset
         metrics_df
```

Out[19]:

	method	var	variable_type	metric	value
0	DirectPred	CLAUDIN_SUBTYPE	categorical	balanced_acc	0.584327
1	DirectPred	CLAUDIN_SUBTYPE	categorical	f1_score	0.738919
2	DirectPred	CLAUDIN_SUBTYPE	categorical	kappa	0.666531
3	DirectPred	CLAUDIN_SUBTYPE	categorical	average_auroc	0.926714
4	DirectPred	CLAUDIN_SUBTYPE	categorical	average_aupr	0.809894

Extracting the sample embeddings

All models trained within `flexynesis` comes with a `transform` method, which extracts the sample embeddings that are generated by the encoding networks (whether it is an MLP or a variational autoencoder). The embeddings reflect a merged representation of multiple omic layers.

```
In [20]: ds = test_dataset
         E = model.transform(ds)
```

```
In [21]: E.head()
```

Out[21]:

	E0	E1	E2	E3	E4	E5	E6	E7	E8	E9	...	
MB-5049	-0.244987	-0.016129	-0.316997	0.168778	0.061209	0.270115	0.281651	0.241118	0.814357	-0.011755	...	-0.4
MB-5477	-0.030940	0.021014	-0.115347	-0.015015	-0.034387	0.038424	0.029271	-0.130029	-0.133066	-0.231606	...	-0.6
MB-4001	-0.171398	0.029773	-0.130385	-0.086179	-0.172552	-0.072600	0.372836	-0.333574	0.234509	-0.368763	...	0.3
MB-0517	-0.137529	0.026078	-0.057952	-0.001665	-0.205516	-0.190910	0.364182	-0.458751	0.234838	-0.164069	...	0.1
MB-6050	-0.140784	-0.176784	-0.094458	0.308385	-0.036618	0.458884	0.408140	-0.058658	0.219353	-0.687358	...	-0.

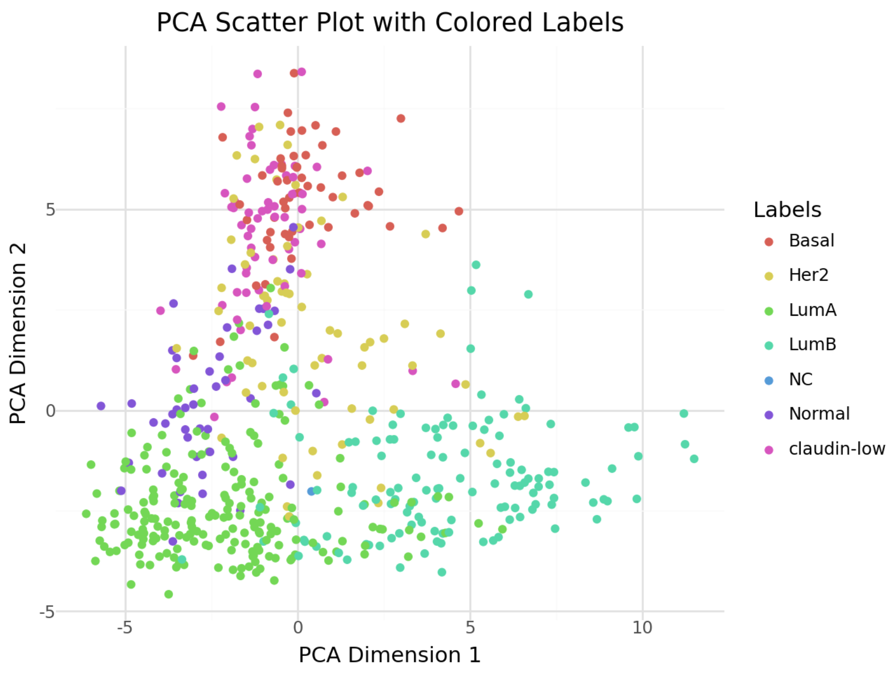
5 rows × 164 columns

Visualizing the sample embeddings

Let's visualize the embeddings in a reduced space and color by the target variables.

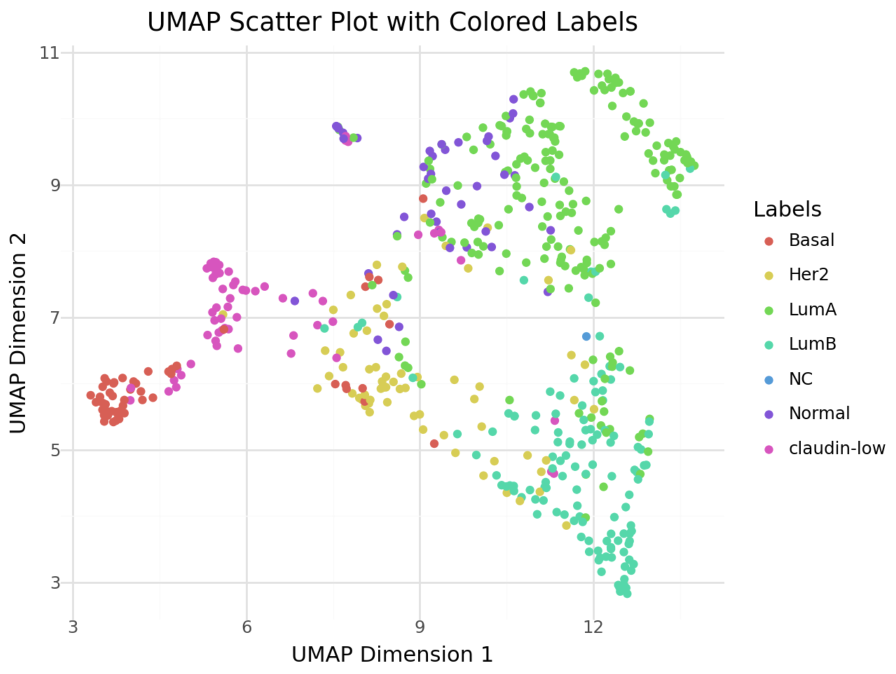
```
In [22]: f = 'CLAUDIN_SUBTYPE'
         labels = [ds.label_mappings[f][x] for x in ds.ann[f].numpy()] #map the sample labels from numeric vector to ini
```

```
In [23]: flexynesis.plot_dim_reduced(E, labels, color_type = 'categorical', method='pca')
```



We can also use UMAP visualisation

```
In [24]: flexynesis.plot_dim_reduced(E, labels, color_type = 'categorical', method='umap')
```



```
In [ ]:
```