# Model

# Reproduce Models

### Data setup for the models

Make sure you download all the datasets before starting to train. The datasets are provided as CSV files to the training pipelines with `image_path` column providing location of the image to be used, `coordX`, `coordY` and `coordZ` providing the global coordinates of the seed point around which a patch is cropped. We crop a [50, 50, 50] patch around the seed point. Please refer to our paper for more details on this. Along with these columns, label columns are needed for supervised training. Labels for each task are as follows,

```
Task 1: Coarse_lesion_type
Task 2: malignancy
Task 3: survival
```

### Reproducing our foundation model

The crux of our study is the self-supervised training procedure. We implemented contrastive pre-training using a modified version of the SimCLR framework. The SimCLR framework's general principle involves transforming a single data piece (e.g., a patch taken from a CT scan) into two correlated and augmented samples (e.g., the same patch rotated 15 degrees clockwise and flipped horizontally). A convolutional encoder is then used to extract latent representations from these samples. Through a contrastive loss function, the model learns to identify similar representations from the same data sample and dissimilar representations from different data samples. The framework emphasizes effective transformation choices, convolutional encoder architectures, and contrastive loss functions for optimal self-supervised learning performance. To effectively represent the nature of medical images, we made modifications to each of these components.

1. Medical image specific transformations implemented from Project-MONAI (https://monai.io/) and custom implementations at `fmcib.ssl.transforms`
2. 3D ResNet from Project-MONAI
3. Custom implemented modified loss function and SimCLR architecture that can be found under `fmcib.ssl.losses.NTXentNegativeMinedLoss` and `fmcib.ssl.models.ExNegSimCLR`

We use project-lighter developed internally within our lab to provide reproducible training for all the models used in this study. Project-lighter allows a YAML-based configuration system along with a python-based CLI to allow quick, easy and scalable experimentation.

To pre-train on the DeepLesion pretraining set, you can find the YAML for the pre-training at `experiments/pretraining/simclr_pretrain.yaml`. It is assumed that you have a GPU available. If you do not (not recommended and not tested), then edit the following parameters

```
accelerator: cpu
strategy: auto
devices: 1
```

The default training assumes 2 GPUs as mentioned in the paper. You can change this by setting

```
devices: 1
```

Change the path of the train dataset to the pre-train set generated earlier in the data-preprocessing step

```
train_dataset:
  _target_: fmcib.ssl.datasets.SSLRadiomicsDataset
  path: "your_pretrain_set_path_goes_here"
```

Now you can start training by running this in the root code folder,

```
lighter fit --config_file ./experiments/pretraining/simclr_pretrain.yaml
```

## Reproducing our supervised training

As mentioned in section, we have three different supervised training implementations. Similar to the foundation pre-training, we use YAML files to maintain the configurations of these implementations.

#### Supervised model trained from random initialization

In order to reproduce this training, you can inspect the YAML configuration at `experiments/supervised_training/supervised_random_init.yaml`. By default, we configure this for Task 1. You can adapt this for Task 2 and Task 3 by searching for 'Note: ' comments in the YAML that outline what must be changed.

You can start training by running this in the root code folder,

```
lighter fit --config_file ./experiments/supervised_training/supervised_random_init.yaml
```

#### Fine-tuning a trained supervised model

The YAML configuration at `experiments/supervised_training/supervised_finetune.yaml` describes how you can fine-tune an already trained supervised model. Note that this is possible only for Task 2 and Task 3 as we used the supervised model trained in Task 1 to load weights from. Make sure you download the weights for Task 1 supervised models. You can follow instructions here

You can start training by running this in the root code folder,

```
lighter fit --config_file ./experiments/supervised_training/supervised_finetune.yaml
```

**Fine-tuning a pre-trained foundation model**

We provide the YAML configuration for this at `experiments/supervised_training/foundation_finetune.yaml`. Similar to the random initialization supervised training, it has been configured for Task 1 with 'Note:' tags for modifying to other tasks. Make sure you download the weights for the pre-trained foundation model before attempting to reproduce this training. You can follow instructions here

You can start training by running this in the root code folder,

```
lighter fit --config_file ./experiments/supervised_training/foundation_finetune.yaml
```

## Reproducing our linear evaluation (Logistic Regression)

We extracted 4096 features from the foundation model for each data point and used them to train a logistic regression model using the scikit-learn framework. A comprehensive parameter search for the logistic regression model was performed using the optuna hyper-parameter optimization framework. The code and utilities for performing the logistic regression modelling is provided in `experiments/linear_evaluation`

In order to perform the modelling, you can run

```
python run.py <features_folder> <label>
```

The <features_folder> must contain `train_features.csv`, `val_features.csv` and `test_features.csv` all extracted from our foundation model. The process to extract features from our foundation model is highlighted in this section

The corresponds to the column in the csv files that contains the supervised label to predict. For example, in use-case 2 the label is `malignancy`.

You can also provide scoring metrics, for instance, using `--scoring roc_auc` where the scoring metric is a sklearn scorer. You can also provide the number of trials the optimization framework needs to be run for using `--trials`.

The features folder is provided under `outputs/foundation_features` to try our the modelling process. Refer here