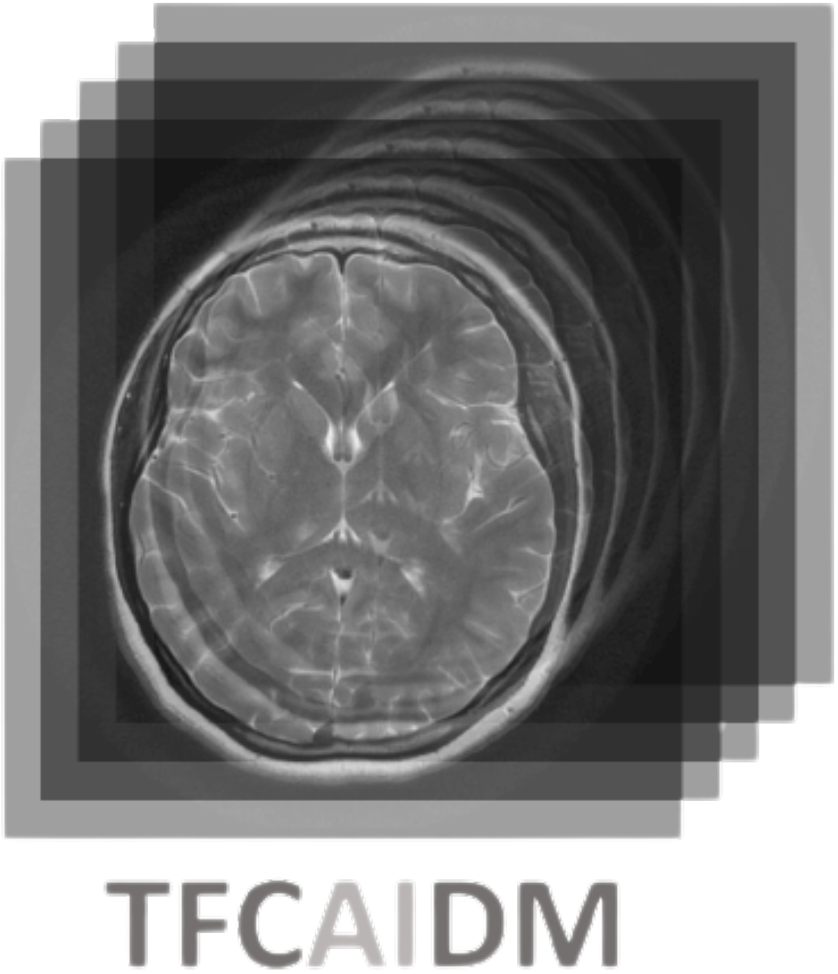
TFCAIDM

TensorFlow CAIDM

Deep learning pipeline for medical imaging

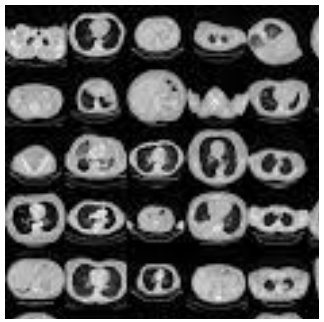
github pypi

December 2, 2021

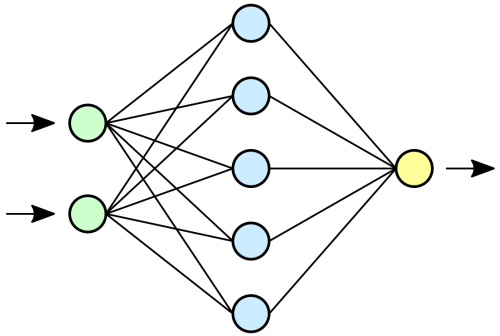


Introduction

TFCAIDM is a unified framework for building and training medical imaging deep learning models built on top of TensorFlow and JarvisMD. The library supports interfacing custom datasets with jarvis, model development with tensorflow, and built-in reproducibility, traceability, and performance logging for all experiments.

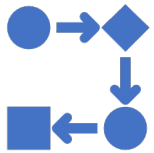


jarvis



tensorflow

Motivation



To make it easier to iterate over different ideas and get results.

To make it easier to keeping track of different experiments.

To make it easier to compare and reproduce results.

A standard way to do things

Many machine learning projects are similar in principle. Though specific implementations may differ depending on factors such as the task being solved, dataset size, class balance, label quality, etc., there should be a way to automate the entire pipeline such that training different algorithms on different datasets is not so manual and tedious.

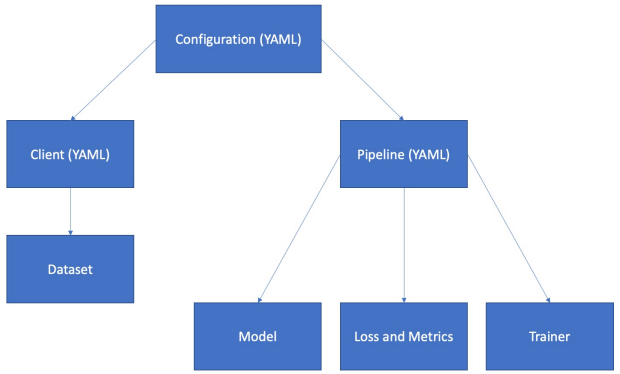
Features

- Reusable models
- Hyperparameter tuning
- Reproducible results
- Experiment tracking
- Checkpointing and logging

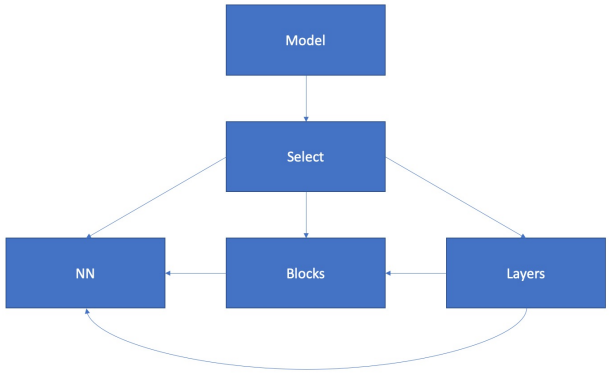
The library can run an entire machine learning project straight from a YAML file.

Additionally, many portions of the library are easily (hopefully) customizable and extendable.

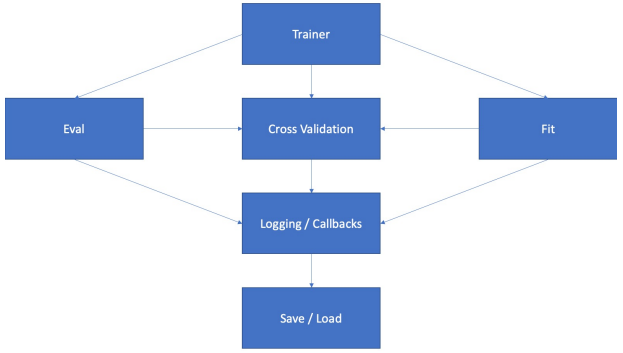
Config



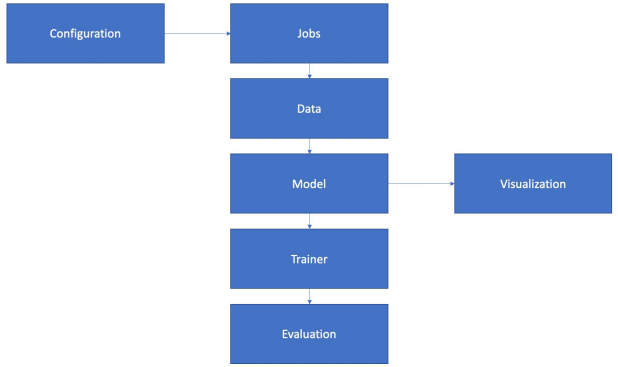
Model



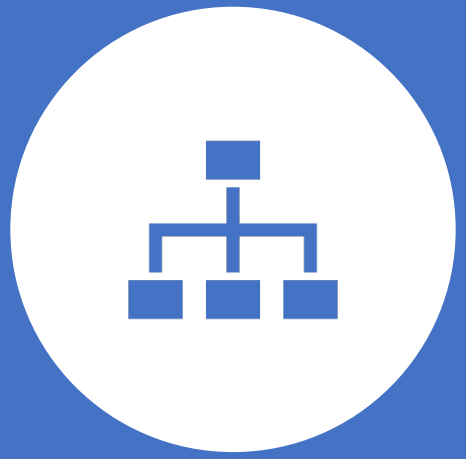
Trainer



Pipeline



Methodology



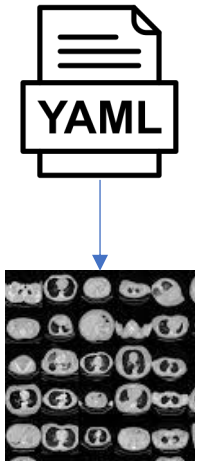
Modules

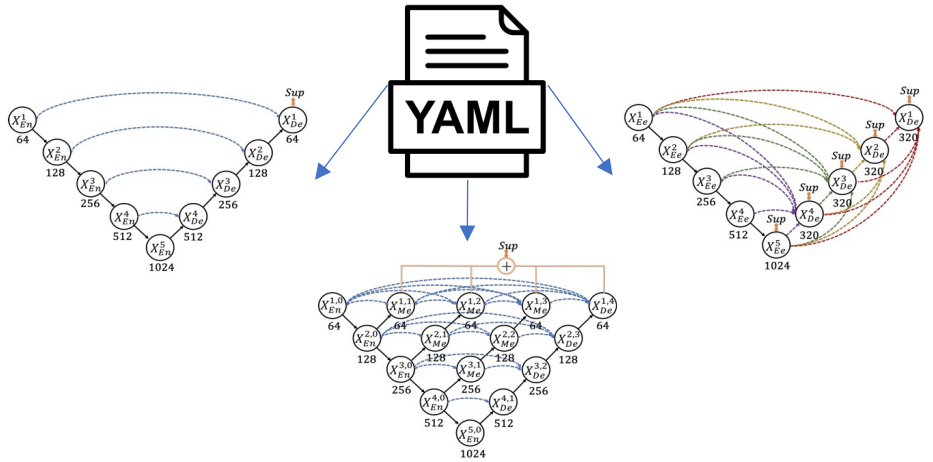
Config, Jobs, Data, Models, Loss and Metrics, Trainer, Tools

Config

Everything is setup based on two YAML configuration files

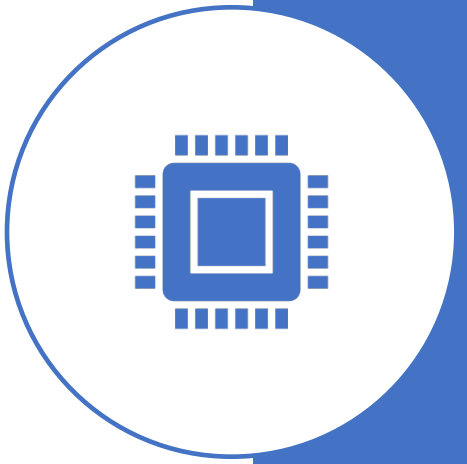
- client.yml: configure dataset
- pipeline.yml: configure model and training





Jobs

For submitting training routines to the GPU cluster.



```python
from jarvis.utils.general import gpus
from tfcaidm import Jobs

# --- Define paths
YML_CONFIG = "pipeline.yml"
TRAIN_ROUTINE_PATH = "main.py"

# --- Submit a training job
Jobs(path=YML_CONFIG).setup(
    producer=__file__,
    consumer=TRAIN_ROUTINE_PATH,
).train_cluster()
```

submit.py

```python
from jarvis.train import params
from jarvis.utils.general import gpus
from tfcaidm import Trainer

# --- Autoselect GPU (use only on caidm cluster)
gpus.autoselect()

# --- Get hyperparameters (args passed by environment variables)
hyperparams = params.load()

# --- Train model (dataset and model created within trainer)
trainer = Trainer(hyperparams)
results = trainer.cross_validation(save=True)
trainer.save_results(results)
```

main.py

Data

Handles data loading, preprocessing, and splitting. The bulk functionality is handled by JarvisMD with some added features for automation and preprocessing extended by **TFCAIDM.**

All data related things are handled through the client.yml file with the option to add custom functionality through overloading the client class.

For more on the jarvis client see:

- https://github.com/peterchang77/dl_tutor/tree/master/jarvis
- https://github.com/Brandhsu/tfcaidm/blob/master/tfcaidm/data/jclient.py

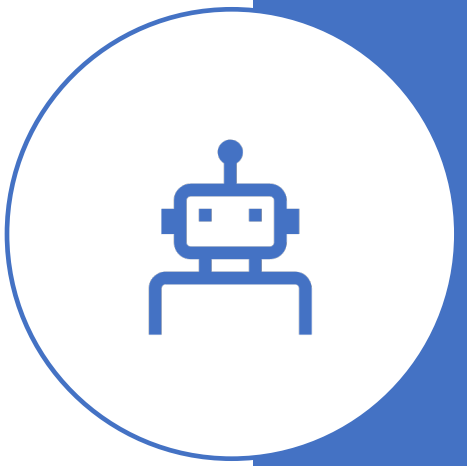
Models

Various model architectures and blocks are implemented and useable out-of-the-box with compatibility for both 2D and 3D data.

Some models include UNet, UNet++, and UNet3+.

Some model blocks include dense connections, squeeze-excitation, convolutional attention, and atrous spatial pyramid pooling.

Several output heads are also implemented such as deep and multiscale supervision.



Loss and Metrics

Popular loss functions for both classification and segmentation are implemented such as cross-entropy, focal loss, dice loss, etc.

All losses can automatically get assigned class weights through the pipeline.yml file.

Several training metrics are also defined such as balanced accuracy and F1 score.

Trainer

The trainer comes with methods for training, evaluation, and cross validation.

The trainer utilizes standard TensorFlow callbacks for tensorboard logging, model checkpointing, learning rate decay, etc.

Additionally, the trainer saves the entire result of model training with all hyperparameter settings in CSV and YAML files for experimental tracking and reproducibility purposes.

Tools

- Some handy tools that are implemented include tensorboard viewing, comparing experiments, and deleting experiments.

Examples

Configuration, Customization, Viewing Results

Configuration

https://github.com/Brandhsu/tfcaidm/tree/master/configs

Customization

For another day, still need to write some documentation.

☺☺☺☺

☺☺

☺

Viewing Results

- Look at those experiments, so many, how can I checkout the best few models quickly?

- Pandas + Statistics = Profit

Microsoft gave me a panda design how cute.

Closing Remarks

Next Steps, Contributing

Next Steps

- AutoML / efficient hyperparameter search
- Distributed data and model training
- Vision transformer models
- Benchmarks

Contributing

Eventually the models we build need to be supported on robust infrastructure and tested in a standard and reproducible manner, this acts as a steppingstone to that goal.

Building reusable tools is awesome

Helps others (and yourself) work more efficiently

Gain valuable skills as a software developer and researcher

https://github.com/Brandhsu/tfcaidm | https://pypi.org/project/tfcaidm/