1.train.py

```
from self_supervised_3d_tasks.data.numpy_2d_loader import Numpy2DLoader
from self_supervised_3d_tasks.utils.model_utils import init, print_flat_summary
from pathlib import Path

import tensorflow.keras as keras
from self_supervised_3d_tasks.data.numpy_3d_loader import
DataGeneratorUnlabeled3D, PatchDataGeneratorUnlabeled3D

from self_supervised_3d_tasks.data.make_data_generator import
get_data_generators
from self_supervised_3d_tasks.data.image_2d_loader import
DataGeneratorUnlabeled2D
from self_supervised_3d_tasks.algorithms import cpc, jigsaw,
relative_patch_location, rotation, exemplar
from self_supervised_3d_tasks.utils.model_utils import get_writing_path

keras_algorithm_list = {
    "cpc": cpc,
    "jigsaw": jigsaw,
    "rpl": relative_patch_location,
    "rotation": rotation,
    "exemplar": exemplar
}

data_gen_list = {
    "kaggle_retina": DataGeneratorUnlabeled2D,
    "pancreas3d": DataGeneratorUnlabeled3D,
    "pancreas2d": Numpy2DLoader,
    "brats": DataGeneratorUnlabeled3D,
    "ukb2d": DataGeneratorUnlabeled2D,
    "ukb3d": PatchDataGeneratorUnlabeled3D
}


def get_dataset(data_dir, batch_size, f_train, f_val, train_val_split,
dataset_name,
                train_data_generator_args={}, val_data_generator_args={},
**kwargs):
    data_gen_type = data_gen_list[dataset_name]

    train_data, validation_data = get_data_generators(data_dir,
train_split=train_val_split,
                                                      train_data_generator_args=
{**{"batch_size": batch_size,

   "pre_proc_func": f_train},

**train_data_generator_args},
                                                      val_data_generator_args=
{**{"batch_size": batch_size,

 "pre_proc_func": f_val},
```

```
**val_data_generator_args},

 data_generator=data_gen_type)

    return train_data, validation_data


def train_model(algorithm, data_dir, dataset_name, root_config_file, epochs=250,
batch_size=2, train_val_split=0.9,
                base_workspace="~/netstore/workspace/",
save_checkpoint_every_n_epochs=5, **kwargs):
    kwargs["root_config_file"] = root_config_file

    working_dir = get_writing_path(Path(base_workspace).expanduser() /
(algorithm + "_" + dataset_name),
                                   root_config_file)
    algorithm_def = keras_algorithm_list[algorithm].create_instance(**kwargs)

    f_train, f_val = algorithm_def.get_training_preprocessing()
    train_data, validation_data = get_dataset(data_dir, batch_size, f_train,
f_val, train_val_split, dataset_name, **kwargs)
    model = algorithm_def.get_training_model()
    print_flat_summary(model)

    tb_c = keras.callbacks.TensorBoard(log_dir=str(working_dir))
    mc_c = keras.callbacks.ModelCheckpoint(str(working_dir / "weights-
improvement-{epoch:03d}.hdf5"), monitor="val_loss",
                                           mode="min", save_best_only=True)  #
reduce storage space
    mc_c_epochs = keras.callbacks.ModelCheckpoint(str(working_dir / "weights-
{epoch:03d}.hdf5"), period=save_checkpoint_every_n_epochs)  # reduce storage
space
    callbacks = [tb_c, mc_c, mc_c_epochs]

    # Trains the model
    model.fit_generator(
        generator=train_data,
        steps_per_epoch=len(train_data),
        validation_data=validation_data,
        validation_steps=len(validation_data),
        epochs=epochs,
        callbacks=callbacks
    )


def main():
    init(train_model)


if __name__ == "__main__":
    main()
```