Training Manager Configuration

Before making any run we have to configure the training manager script. This documentation will be a guide to do so with a walk through to the process.

Step 1: Required Function Imports:

First of all we have to import our model and necessary function to run the training. Here is the screenshot of a section of the training manager.

```
1  import tensorflow as tf
2  import numpy as np
3
4  #Adding the default path to the data directory
5  default_dataset_directory='GeometryUtilities-master/interpolation/image_data/'
6
7  #importing the model to be used for training
8  from models.model_rnn_definition import model9 as model_function_handle
9  from models.model1_definition import calculate_model_accuracy
10 from models.model_rnn_definition import calculate_total_loss
11
12 #import the trainer and inference functions
13 from train_multi_gpu import train
14 from inference_multi_gpu import infer
15 #import the gradient calulation function
16 from get_saliency_map import get_gradient
17
```

1. **Line 5:** Here we could specify the default dataset directory where our training and inference dataset is present. This could also be given by the command line argument.
2. **Line 8:** Here we have to import the model from the **model.model_xyz_definition** script which we had defined in the **models/** folder. For example here the model9 is being currently being used from the model_rnn_definition script.
3. **Line 9 and 10:** Here we have to specify which loss function and accuracy calculation function we want to use for this model.
4. **Line 13-16:** this will remain same throughout

Step 2: Required Function Imports:

Now we will specify the training-**"run"** configuration like the run number and dataset file.

```
18 ##################### RUN CONFIGURATION ####################
19 run_number=47
20 #the regex pattern for the dataset filename
21 train_filename_pattern='nopu/train/small/*'
22 test_filename_pattern='nopu/valid/small/*'
23 test_pu_filename_pattern='test_pu/*'
24 viz_filename_pattern='test_pu/*'
25
26 if __name__=='__main__':
```

1. **Line 19:** We have to give a unique run number for each unique run to save the corresponding results and tensorboard files.
2. **Line 21: train_filename_pattern** takes the regex pattern for the files to be included in the training dataset. This will be used internally by tf.data to match the files matching this pattern to automatically include them in the training dataset.
3. Similarly for the other filename we could provide the directory pattern to get the data from.

Step 3:Configuring Training Arguments:

Given we are using the command line argument --mode=train, we have to configure these section. In the screenshot below please see the refereed line number.

1. **init_learning_rate: (line 64)** This will be the initial learning rate for the model to be used by the optimizer.
2. **decay_rate and decay_step: (line 65-66)** This will be used for applying the exponential decay to the learning rate by **multiplying the learning rate with the factor of decay_rate after every decay_step**. For more information please see here. https://www.tensorflow.org/api_docs/python/tf/train/exponential_decay
3. **mini_batch_size:** The number of image to process in parallel. For current CNN models **minibatch size of 20** is efficient (both time and memory wise) and **10** for the RNN Modules
4. **shuffle_buffer_size:** this is used to shuffle the tfrecords (files) in the dataset.
5. **epochs:** the number of epoch we want to run the training.
6. **restore_epoch_number:** for restoring the training from the saved checkpoint of parameters in the model instead of starting again.

That's all is needed for the starting the training.

```
################# TRAINING HANDLE ####################
'''
DESCRIPTION:
    This is the main control of all the training and the hyperparameter
    definition of the training.
    All the tensorboard visualization and the checkpoints of the model
    will be saved in the current directory under the directory structure:

    current_directory/tmp/hgcal/run_number

    This direcotry is by default added to the gitignore
'''
if opt.mode=='train':
    #Specifying the Hyperparameters
    init_learning_rate=0.001
    decay_step=60
    decay_rate=0.95
    #Specifying the run configuration
    mini_batch_size=10
    shuffle_buffer_size=mini_batch_size*2 #for shuffling the dataset files
    epochs=31
    restore_epoch_number=None

    #Finally running the training
    train(run_number,
            model_function_handle,
            calculate_model_accuracy,
            calculate_total_loss,
            epochs,mini_batch_size,shuffle_buffer_size,
            init_learning_rate,decay_step,decay_rate,
            train_filename_pattern,test_filename_pattern,
            restore_epoch_number=restore_epoch_number)
```

Step 4:Configuring Inference Arguments:

Now this function will run the inference on the training and testing data or any data of our choice as specified above in run_configuration section.The arguments we have to provide are:

1. **mini_batch_size:** the number of images we want to process in parallel
2. **checkpoint_epoch_number**: this will specify the checkpoint number (saved parameter of that epoch) to run the inference. One good use of it is that, we could see the variation in the prediction or other visualization as the training progress with different epochs.

That's all is needed for running the inference.

```python
############## INFERENCE HANDLE ######################
'''
DESCRIPTION:
    This is the main point of control for all the inferece task like
    calulating the average accuracy on the training and test/dev set
    for furthur visualization.

    All the results from the current inference will be saved in same
    directory strucute as train module:

    current_directory/tmp/hgcal/run_number
'''
#specifying the inference configuration
if opt.mode=='infer':
    mini_batch_size=10
    checkpoint_epoch_number=31

    #Running the inference on the training data set
    infer(run_number,
            model_function_handle,
            calculate_model_accuracy,
            calculate_total_loss,
            train_filename_pattern,
            inference_mode='train',#on the training dataset
            mini_batch_size=mini_batch_size,
            checkpoint_epoch_number=checkpoint_epoch_number)

    #Now rerunning the inference on the test dataset
    tf.reset_default_graph()
```

Step 5:Configuring for Prediction Visualization:

If we want to see the prediction statistics like the error histogram and other profile histograms, then we could use this part.

This part doesn't have any changeable parameters before use. But it assumes that the inference has run already (and data is saved by it in tmp/hgcal/run_number folder).

```
    ############# Visulaization Handle ###############
    '''
    Description:
        This will be the main point of control for making all the
        visualization of the training including the currently developed
        visualizations like
            1. prediction visulaization (includes error histograms)
            2. saliency maps (the gradient maps giving the sensitive
                                    regions of the prediction in image)
            other visualization will be added later

        This manager will use the results saved by the inference module
        in tmp/hgcal/run_number/results to make the visualization
    '''
    if opt.mode=='pred_viz':
        #Inporting the modules for visualization process
        from Visualization_Module.prediction_visualization import plot_histogram,load_data
        #################### Histogram Plots ###################
        #Loading the data
        filename='tmp/hgcal/{}/results/results_mode_train.npz'.format(run_number)
        train_results=load_data(filename)
        predictions=train_results['predictions']
        labels=train_results['labels']
        #Plotting the histogram
        plot_histogram(predictions,labels)

        #plotting the test set prediction histograms
        filename='tmp/hgcal/{}/results/results_mode_test_pu.npz'.format(run_number)
        test_results=load_data(filename)
        predictions=test_results['predictions']
        labels=test_results['labels']
        #Plotting the histogram
        plot_histogram(predictions,labels)
```

Step 6:Configuring for Saliency Map Visualization:

Now if we want to visualize the saliency map for the current training run, then first we will have to generate the map. For generation it needs following arguments

1. **checkpoint_epoch_number:** again the same argument to say from which saved epoch state of parameters we want to generate the weights.
2. **map_dimension** : The current output format of models is
    a. Energy
    b. Pos-x of barycenter
    c. Pos-y of barycenter
    d. Pos-z of barycenter
    e. Is particle Electron (1 here if yes)
    f. Is particle Photon (1 here is yes)

So we have to specify here with respect to which output dimension we want to see the gradient of input image.
Here the map-dimension = 0 means the gradient is **d(energy)/d(input_image).**

3. **mini_batch_size:** how many gradient maps we want to produce in parallel. Please use just 1 since there are some issue with the higher number as mentioned there.
4. **file_name:** the name which we want to give to the generated saliency map for saving purpose.

```
if opt.mode=='map_gen':
    #################### Saliency Map ####################
    #Creating the saliency map
    checkpoint_epoch_number=9
    #Choosing wrt which output dimension we want to calculate gradient
    map_dimension=0
    #Number of images we want to process in parallel
    '''
    Currently only minibatch size of 1 is supported for calculation of
    gradient. For more information follow this thread.
    https://github.com/tensorflow/tensorflow/issues/4897
    '''
    mini_batch_size=1
    filename='test_pu'
    #Calulating the gradient
    get_gradient(run_number,
                model_function_handle,
                viz_filename_pattern,
                mini_batch_size,
                checkpoint_epoch_number,
                map_dimension,
                filename)

if opt.mode=='map_viz':
```

For visualization purpose there is not extra arguments to change and we could directly visualize the saliency map by selecting the appropriate mode in the command line (--mode=map_viz).