Project Documentation

May 15, 2024

Contents

1 Installation

To set up the project environment, follow these steps:

1. Navigate to the project directory:

   ```
   cd your_repository
   ```

2. Install the required packages using `pip` and the provided `requirements.txt` file:

   ```
   pip install -r requirements.txt
   ```

This command will install all the necessary Python packages specified in the `requirements.txt` file.

Once the required packages are installed, you can proceed to use the project as described in the following sections.

2 Usage

2.1 Training a Model

To train a model, execute the following command in the terminal:

```
python train_model.py --new_checkpoint_name <new_checkpoint_name>
    --load_checkpoint_name <load_checkpoint_name> --epochs <epochs>
```

Listing 1: Training a Model CLI

CLI Options:

- `--new_checkpoint_name, -n`: Specifies the name of the checkpoint to save after training on every epoch. The default name is "model".
- `--load_checkpoint_name, -l`: Specifies the path to the model weights to load before training. By default, no pre-trained weights are loaded.
- `--epochs, -e`: Specifies the number of epochs to train the model.

Example:
To train a model for 100 epochs and save the checkpoint as "my_model", use the following command:

```
python train_model.py -n my_model -e 100
```

Listing 2: Example: Training a Model

This command will train the model for 100 epochs and save the checkpoint as "my_model".

Additionally, users can modify hyperparameters using the JSON configuration file located at `config/hyperparameters.json`. This file allows users to specify hyperparameters such as learning rate, batch size, etc. The value "auto" indicates that the parameter uses the default value, or it is calculated dynamically before training.

2.2 Testing a Model

To test a model using a checkpoint file, run the following command:

```
python test_model.py <checkpoint_name>
```

Listing 3: Testing a Model CLI

Arguments:

- `checkpoint_name`: Specifies the name of the checkpoint file to load for testing.

Example:
To test a model using a checkpoint named "my_model_checkpoint.pth", use the following command:

```
python test_model.py my_model_checkpoint.pth
```

Listing 4: Example: Testing a Model

This command will load the checkpoint file "my_model_checkpoint.pth" located in the models directory and perform testing on the model.

3 Code Documentation

3.1 OpticalFlowExtractor

Source: `video_dataset.py`
This class extracts optical flow using the RAFT optical flow model.

- **Methods**:
  - `__init__(self)`: Initializes the OpticalFlowExtractor with a pre-trained RAFT model.
  - `forward(batch_frames)`: Performs optical flow extraction on a batch of frames.

3.2 CMWRA

Source: `models.py`
This class implements a Convolutional Neural Network (CNN) with motion features extracted by OpticalFlowExtractor.

- **Methods**:
  - `__init__(self, input_shape, dropout_prob=0.5)`: Initializes the CMWRA model with the specified input shape and dropout probability.
  - `forward(x)`: Performs forward pass through the CMWRA model.

3.3 BasicBlock3D

Source: `models.py`
Purpose: This class defines a basic 3D residual block.

- **Methods**:
  - `__init__(self, in_planes, out_planes, stride=1, dropout_prob=0.0)`: Initializes the BasicBlock3D with specified parameters.
  - `forward(x)`: Performs forward pass through the BasicBlock3D.

3.4 CMWRU

Source: `models.py`
Purpose: This class implements a Convolutional Neural Network with Residual Units (CMWRU).

- **Methods**:
  - `__init__(self, input_shape, dropout_prob=0.5)`: Initializes the CMWRU model with the specified input shape and dropout probability.
  - `forward(x)`: Performs forward pass through the CMWRU model.

3.5 Resblock

Source: `models.py`
Purpose: This class defines a residual block with two convolutional layers.

- **Methods**:
  - `__init__(self, shape)`: Initializes the Resblock with the specified shape.
  - `forward(x)`: Performs forward pass through the Resblock.

3.6 RCM2d_Small

Source: `models.py`
Purpose: This class implements a Convolutional Neural Network (CNN) with recurrent connections for 2D data.

- **Methods**:
  - `__init__(self, input_shape, dropout_prob=0.5)`: Initializes the RCM2d_Small model with the specified input shape and dropout probability.
  - `forward(x)`: Performs forward pass through the RCM2d_Small model.

3.7 VideoDataset

Source: `video_dataset.py`
Purpose: This class implements a PyTorch dataset for loading video data.

- **Attributes**:
  - `df`: DataFrame containing video information.
  - `datadir`: Directory containing video data.
  - `data`: List of video names.
  - `targets`: Tensor containing video class labels.

- `transform`: Optional transformation to be applied to video frames.

- **Methods**:
  - `__init__(self, df, datadir, transform=None)`: Initializes the VideoDataset with the provided DataFrame, data directory, and optional transformation.
  - `__len__()`: Returns the length of the dataset.
  - `__getitem__(self, idx)`: Retrieves the item at the specified index, loading and transforming the video frames as necessary.

3.8 FocalLoss

Source: `focal_loss.py`
Purpose: This class implements the Focal Loss function for use in training neural networks.

- **Attributes**:
  - `alpha`: The alpha parameter of the Focal Loss function.
  - `gamma`: The gamma parameter of the Focal Loss function.
  - `reduction`: The reduction method for aggregating the loss values.

- **Methods**:
  - `__init__(self, alpha: float = 0.25, gamma: float = 2, reduction: str = 'none')`: Initializes the FocalLoss with the specified parameters.
  - `forward(self, inputs, targets)`: Computes the Focal Loss given the input predictions and target labels.

3.9 prepare_data

Source: `utils/prepare_data.py`
Purpose: This function prepares the dataset for training by splitting it into train, validation, and test sets, balancing the class distribution, and calculating ratios.

- **Parameters**:
  - `df`: DataFrame containing video information.
  - `sampled_size`: The size of the sampled dataset.
  - `train_test_size`: The size of the train-test split.
  - `train_val_size`: The size of the train-validation split.
  - `seed`: Seed for random number generation.
  - `class_ratio`: Optional ratio for balancing the class distribution.

- **Returns**:
  - List containing train, validation, and test sets, and the calculated class ratio.

3.10 calculate_overall_mean_of_means_and_stds

Source: `utils/calculate_mean_std.py`
Purpose: This function calculates the overall mean of means and standard deviations across all batches of data.

- **Parameters**:
  - `data_loader`: DataLoader containing the dataset.
  - `device`: Device to perform calculations on (e.g., 'cpu', 'cuda').
- **Returns**:
  - List containing the overall mean of means.
  - List containing the overall mean of standard deviations.

3.11 print_history

Source: `utils/print_history.py`
Purpose: This function prints the training/validation history including metrics such as loss, accuracy, precision, recall, F1 score, specificity, confusion matrix, and time.

- **Parameters**:
  - `history`: Dictionary containing training/validation history.
  - `mode`: Mode of history to print ('train' or 'val').
  - `epoch`: Epoch for which to print history.
- **Output**:
  - Prints the history for the specified mode and epoch.

3.12 get_batch_size

Source: `utils/find_batch_size.py`
Purpose: This function determines the optimal batch size for training a given model on a dataset.

- **Parameters**:
  - `model`: The neural network model to be trained.
  - `criterion`: The loss function used for optimization.

  - `optimizer`: The optimization algorithm used for training.
  - `input_shape`: The shape of the input data.
  - `output_shape`: The shape of the output data.
  - `dataset_size`: The total size of the dataset.
  - `device`: The device to perform computations on (e.g., ćpu, ćuda).
  - `max_batch_size`: Optional maximum batch size allowed.
  - `num_iterations`: Number of iterations to perform for each batch size estimation.

- **Returns**:
  - Optimal batch size for training the model.