torch-tensornet

Release 1.3.3

Shantanu Acharya

Nov 27, 2021

CONTENTS

TensorNet is a high-level deep learning library built on top of PyTorch.

CHAPTER

ONE

INSTALLATION

To install and use TensorNet all you have to do is:

```
pip install torch-tensornet
```

If you want to get the latest version of the code before it is released on PyPI you can install the library from GitHub

```
pip install git+https://github.com/shan18/TensorNet.git#egg=torch-tensornet
```

1.1 Data

Classes and methods which can be used to create and modify datasets.

1.1.1 Datasets

class tensornet.data.**BaseDataset**(*train_batch_size=1*, *val_batch_size=1*, *cuda=False*, *num_workers=1*, *path=None*, *train_split=0.7*, *resize=0, 0*, *padding=0, 0*, *crop=0, 0*, *horizontal_flip_prob=0.0*, *vertical_flip_prob=0.0*, *gaussian_blur_prob=0.0*, *rotate_degree=0.0*, *cutout_prob=0.0*, *cutout_dim=8, 8*, *hue_saturation_prob=0.0*, *contrast_prob=0.0*)

Loads a dataset.

Parameters

- **train_batch_size** (int, optional) – Number of images to consider in each batch in train set. (default: 0)
- **val_batch_size** (int, optional) – Number of images to consider in each batch in validation set. (default: 0)
- **cuda** (bool, optional) – True is GPU is available. (default: False)
- **num_workers** (int, optional) – How many subprocesses to use for data loading. (default: 0)
- **path** (str, optional) – Path where dataset will be downloaded. If no path provided, data will be downloaded in a pre-defined directory.
- **train_split** (float, optional) – Fraction of dataset to assign for training. This parameter will not work for MNIST and CIFAR-10 datasets. (default: 0.7)
- **resize** (tuple, optional) – Resize the input to the given height and width. (default: (0, 0))

- **padding** (tuple, optional) – Pad the image if the image size is less than the specified dimensions (height, width). (default: (0, 0))
- **crop** (tuple, optional) – Randomly crop the image with the specified dimensions (height, width). (default: (0, 0))
- **horizontal_flip_prob** (float, optional) – Probability of an image being horizontally flipped. (default: 0)
- **vertical_flip_prob** (float, optional) – Probability of an image being vertically flipped. (default: 0)
- **rotate_degree** (float, optional) – Angle of rotation for image augmentation. (default: 0)
- **cutout_prob** (float, optional) – Probability that cutout will be performed. (default: 0)
- **cutout_dim** (tuple, optional) – Dimensions of the cutout box (height, width). (default: (8, 8))
- **hue_saturation_prob** (float, optional) – Probability of randomly changing hue, saturation and value of the input image. (default: 0)
- **contrast_prob** (float, optional) – Randomly changing contrast of the input image. (default: 0)

data(*train=True*)

Return data based on train mode.

Parameters **train** (bool, optional) – True for training data. (default: True)

Returns Training or validation data and targets.

unnormalize(*image*, *transpose=False*, *data_type=None*)

Un-normalize a given image.

Parameters

- **image** (numpy.ndarray or torch.Tensor) – A ndarray or tensor. If tensor, it should be in CPU.
- **transpose** (bool, optional) – If True, transposed output will be returned. This param is effective only when image is a tensor. If tensor, the output will have channel number as the last dim. (default: False)
- **data_type** (str, optional) – Type of image. Required only when dataset has multiple types of images. (default: None)

Returns Unnormalized image

Return type (*numpy.ndarray* or *torch.Tensor*)

normalize(*image*, *transpose=False*, *data_type=None*)

Normalize a given image.

Parameters

- **image** (numpy.ndarray or torch.Tensor) – A ndarray or tensor. If tensor, it should be in CPU.
- **transpose** (bool, optional) – If True, transposed output will be returned. This param is effective only when image is a tensor. If tensor, the output will have channel number as the last dim. (default: False)

- **data_type** (str, optional) – Type of image. Required only when dataset has multiple types of images. (default: None)

Returns Normalized image

Return type (*numpy.ndarray* or *torch.Tensor*)

loader(*train=True*)
Create data loader.

Parameters **train** (bool, optional) – True for training data. (default: True)

Returns Dataloader instance.

class tensornet.data.**MNIST**(*train_batch_size=1*, *val_batch_size=1*, *cuda=False*, *num_workers=1*, *path=None*, *train_split=0.7*, *resize=0, 0*, *padding=0, 0*, *crop=0, 0*, *horizontal_flip_prob=0.0*, *vertical_flip_prob=0.0*, *gaussian_blur_prob=0.0*, *rotate_degree=0.0*, *cutout_prob=0.0*, *cutout_dim=8, 8*, *hue_saturation_prob=0.0*, *contrast_prob=0.0*)
MNIST Dataset.

Note: This dataset inherits the BaseDataset class.

class tensornet.data.**CIFAR10**(*train_batch_size=1*, *val_batch_size=1*, *cuda=False*, *num_workers=1*, *path=None*, *train_split=0.7*, *resize=0, 0*, *padding=0, 0*, *crop=0, 0*, *horizontal_flip_prob=0.0*, *vertical_flip_prob=0.0*, *gaussian_blur_prob=0.0*, *rotate_degree=0.0*, *cutout_prob=0.0*, *cutout_dim=8, 8*, *hue_saturation_prob=0.0*, *contrast_prob=0.0*)
CIFAR-10 Dataset.

Note: This dataset inherits the BaseDataset class.

class tensornet.data.**CIFAR100**(*train_batch_size=1*, *val_batch_size=1*, *cuda=False*, *num_workers=1*, *path=None*, *train_split=0.7*, *resize=0, 0*, *padding=0, 0*, *crop=0, 0*, *horizontal_flip_prob=0.0*, *vertical_flip_prob=0.0*, *gaussian_blur_prob=0.0*, *rotate_degree=0.0*, *cutout_prob=0.0*, *cutout_dim=8, 8*, *hue_saturation_prob=0.0*, *contrast_prob=0.0*)
CIFAR-100 Dataset.

Note: This dataset inherits the BaseDataset class.

class tensornet.data.**TinyImageNet**(*train_batch_size=1*, *val_batch_size=1*, *cuda=False*, *num_workers=1*, *path=None*, *train_split=0.7*, *resize=0, 0*, *padding=0, 0*, *crop=0, 0*, *horizontal_flip_prob=0.0*, *vertical_flip_prob=0.0*, *gaussian_blur_prob=0.0*, *rotate_degree=0.0*, *cutout_prob=0.0*, *cutout_dim=8, 8*, *hue_saturation_prob=0.0*, *contrast_prob=0.0*)
Tiny ImageNet Dataset.

Note: This dataset inherits the BaseDataset class.

class tensornet.data.**MODESTMuseum**(*train_batch_size=1*, *val_batch_size=1*, *cuda=False*, *num_workers=1*, *path=None*, *train_split=0.7*, *resize=0, 0*, *padding=0, 0*, *crop=0, 0*, *horizontal_flip_prob=0.0*, *vertical_flip_prob=0.0*, *gaussian_blur_prob=0.0*, *rotate_degree=0.0*, *cutout_prob=0.0*, *cutout_dim=8, 8*, *hue_saturation_prob=0.0*, *contrast_prob=0.0*)
MODEST Museum Dataset.

Note: This dataset inherits the BaseDataset class.

1.1.2 Processing

class tensornet.data.processing.**Transformations**(*resize=0, 0*, *padding=0, 0*, *crop=0, 0*, *horizontal_flip_prob=0.0*, *vertical_flip_prob=0.0*, *gaussian_blur_prob=0.0*, *rotate_degree=0.0*, *cutout_prob=0.0*, *cutout_dim=8, 8*, *hue_saturation_prob=0.0*, *contrast_prob=0.0*, *mean=0.5, 0.5, 0.5*, *std=0.5, 0.5, 0.5*, *normalize=True*, *train=True*)

Wrapper class to pass on albumentaions transforms into PyTorch.

__call__(*image*)

Process and image through the data transformation pipeline.

Parameters **image** – Image to process.

Returns Transformed image.

Return type (*torch.Tensor*)

tensornet.data.processing.**data_loader**(*data*, *shuffle=True*, *batch_size=1*, *num_workers=1*, *cuda=False*)

Create data loader

Parameters

- **data** (*torchvision.datasets*) – Downloaded dataset.
- **shuffle** (bool, optional) – If True, shuffle the dataset. (default: True)
- **batch_size** (int, optional) – Number of images to considered in each batch. (default: 1)
- **num_workers** (int, optional) – How many subprocesses to use for data loading. (default: 1)
- **cuda** (bool, optional) – True is GPU is available. (default: False)

Returns DataLoader instance.

Return type (*torch.utils.data.DataLoader*)

class tensornet.data.processing.**InfiniteDataLoader**(*data_loader*, *auto_reset=True*)

Create infinite loop in a data loader.

Parameters

- **data_loader** (*torch.utils.data.DataLoader*) – DataLoader object.
- **auto_reset** (bool, optional) – Create an infinite loop data loader. (default: True)

get_batch()

Load next batch from the dataset.

1.2 Models

Classes and methods which can be used to create and configure model architectures.

1.2.1 Base Model

class tensornet.models.**BaseModel**

This is the parent class for all the models that are to be created using `TensorNet`.

forward(*x: torch.Tensor*) → torch.Tensor

This function defines the forward pass of the model.

Parameters **x** (*torch.Tensor*) – Input.

Returns Model output.

Return type (*torch.Tensor*)

summary(*input_size: Tuple[int]*)

Generates model summary.

Parameters **input_size** (*tuple*) – Size of input to the model.

create_learner(*train_loader*, *optimizer*, *criterion*, *device='cpu'*, *epochs=1*, *l1_factor=0.0*, *val_loader=None*, *callbacks=None*, *metrics=None*, *activate_loss_logits=False*, *record_train=True*)

Create Learner object.

Parameters

- **train_loader** (*torch.utils.data.DataLoader*) – Training data loader.
- **optimizer** (*torch.optim*) – Optimizer for the model.
- **criterion** (*torch.nn*) – Loss Function.
- **device** (`str` or `torch.device`) – Device where the data will be loaded.
- **epochs** (`int`, optional) – Numbers of epochs to train the model. (default: 1)
- **l1_factor** (`float`, optional) – L1 regularization factor. (default: 0)
- **val_loader** (`torch.utils.data.DataLoader`, optional) – Validation data loader.
- **callbacks** (`list`, optional) – List of callbacks to be used during training.
- **track** (`str`, optional) – Can be set to either *'epoch'* or *'batch'* and will store the changes in loss and accuracy for each batch or the entire epoch respectively. (default: *'epoch'*)
- **metrics** (`list`, optional) – List of names of the metrics for model evaluation.

set_learner(*learner: tensornet.engine.learner.Learner*)

Assign a learner object to the model.

Parameters **learner** (`Learner`) – Learner object.

fit(*train_loader*, *optimizer*, *criterion*, *device='cpu'*, *epochs=1*, *l1_factor=0.0*, *val_loader=None*, *callbacks=None*, *metrics=None*, *activate_loss_logits=False*, *record_train=True*, *start_epoch=1*, *verbose=True*)

Train the model.

Parameters

- **train_loader** (*torch.utils.data.DataLoader*) – Training data loader.
- **optimizer** (*torch.optim*) – Optimizer for the model.
- **criterion** (*torch.nn*) – Loss Function.
- **device** (str or torch.device) – Device where the data will be loaded.
- **epochs** (int, optional) – Numbers of epochs to train the model. (default: 1)
- **l1_factor** (float, optional) – L1 regularization factor. (default: 0)
- **val_loader** (torch.utils.data.DataLoader, optional) – Validation data loader.
- **callbacks** (list, optional) – List of callbacks to be used during training.
- **track** (str, optional) – Can be set to either *'epoch'* or *'batch'* and will store the changes in loss and accuracy for each batch or the entire epoch respectively. (default: *'epoch'*)
- **metrics** (list, optional) – List of names of the metrics for model evaluation.
- **record_train** (bool, optional) – If False, metrics will be calculated only during validation. (default: True)
- **activate_loss_logits** (bool, optional) – If True, the logits will first pass through the *activate_logits* function before going to the criterion. (default: False)
- **start_epoch** (int, optional) – Starting epoch number to display during training. (default: 1)
- **verbose** (bool, optional) – Print loss and metrics. (default: True)

rfit (*start_epoch=1*, *epochs=None*, *verbose=True*)

evaluate (*loader*, *verbose=True*, *log_message='Evaluation'*)

Evaluate the model on a custom data loader.

Parameters

- **loader** (*torch.utils.data.DataLoader*) – Data loader.
- **verbose** (bool, optional) – Print loss and metrics. (default: True)
- **log_message** (*str*) – Prefix for the logs which are printed at the end.

Returns loss and metric values

save (*filepath: str*, ***kwargs*)

Save the model.

Parameters

- **filepath** (*str*) – File in which the model will be saved.
- ****kwargs** – Additional parameters to save with the model.

load (*filepath: str*) → dict

Load the model and return the additional parameters saved in in the checkpoint file.

Parameters **filepath** (*str*) – File in which the model is be saved.

Returns Parameters saved inside the checkpoint file.

Return type (*dict*)

training: bool

1.2.2 ResNet

class tensornet.models.**ResNet**(*block: Type[Union[tensornet.models.resnet.BasicBlock, tensornet.models.resnet.Bottleneck]]*, *layers: List[int]*, *num_classes: int = 1000*, *zero_init_residual: bool = False*, *groups: int = 1*, *width_per_group: int = 64*, *replace_stride_with_dilation: Optional[List[bool]] = None*, *norm_layer: Optional[Callable[[...], torch.nn.modules.module.Module]] = None*)

Residual-Net (ResNet)

Note: This model inherits the `BaseModel` class.

Parameters

- **block** (`BasicBlock` or `Bottleneck`) – Type of block to use for the model.
- **layers** (*list*) – Number of blocks for each layer.
- **num_classes** (`int`, optional) – Number of classes. (default: 1000)
- **zero_init_residual** (`bool`, optional) – Make residual branch behave like an identity. (default: False)
- **groups** (`int`, optional) – Number of groups per block. (default: 1)
- **width_per_group** (`int`, optional) – Width for each group. (default: 64)
- **replace_stride_with_dilation** (`list` , optional) – Replace stride with dilation for each layer.
- **norm_layer** (`nn.Module` , optional) – Normalization Layer.

forward(*x: torch.Tensor*) → torch.Tensor

This function defines the forward pass of the model.

Parameters **x** (*torch.Tensor*) – Input.

Returns Model output.

Return type (*torch.Tensor*)

training: bool

tensornet.models.**resnet18**(*pretrained: bool = False*, *progress: bool = True*, ***kwargs: Any*) → tensornet.models.resnet.ResNet

ResNet-18 model from "Deep Residual Learning for Image Recognition".

Parameters

- **pretrained** (*bool*) – If True, returns a model pre-trained on ImageNet
- **progress** (*bool*) – If True, displays a progress bar of the download to stderr

tensornet.models.**resnet34**(*pretrained: bool = False*, *progress: bool = True*, ***kwargs: Any*) → tensornet.models.resnet.ResNet

ResNet-34 model from "Deep Residual Learning for Image Recognition".

Parameters

- **pretrained** (*bool*) – If True, returns a model pre-trained on ImageNet
- **progress** (*bool*) – If True, displays a progress bar of the download to stderr

tensornet.models.**resnet50**(*pretrained: bool = False*, *progress: bool = True*, ***kwargs: Any*) → tensornet.models.resnet.ResNet

ResNet-50 model from "Deep Residual Learning for Image Recognition".

Parameters

- **pretrained** (*bool*) – If True, returns a model pre-trained on ImageNet
- **progress** (*bool*) – If True, displays a progress bar of the download to stderr

tensornet.models.**resnet101**(*pretrained: bool = False*, *progress: bool = True*, ***kwargs: Any*) → tensornet.models.resnet.ResNet

ResNet-101 model from "Deep Residual Learning for Image Recognition".

Parameters

- **pretrained** (*bool*) – If True, returns a model pre-trained on ImageNet
- **progress** (*bool*) – If True, displays a progress bar of the download to stderr

tensornet.models.**resnet152**(*pretrained: bool = False*, *progress: bool = True*, ***kwargs: Any*) → tensornet.models.resnet.ResNet

ResNet-152 model from "Deep Residual Learning for Image Recognition".

Parameters

- **pretrained** (*bool*) – If True, returns a model pre-trained on ImageNet
- **progress** (*bool*) – If True, displays a progress bar of the download to stderr

tensornet.models.**resnext50_32x4d**(*pretrained: bool = False*, *progress: bool = True*, ***kwargs: Any*) → tensornet.models.resnet.ResNet

ResNeXt-50 32x4d model from "Aggregated Residual Transformation for Deep Neural Networks".

Parameters

- **pretrained** (*bool*) – If True, returns a model pre-trained on ImageNet
- **progress** (*bool*) – If True, displays a progress bar of the download to stderr

tensornet.models.**resnext101_32x8d**(*pretrained: bool = False*, *progress: bool = True*, ***kwargs: Any*) → tensornet.models.resnet.ResNet

ResNeXt-101 32x8d model from "Aggregated Residual Transformation for Deep Neural Networks".

Parameters

- **pretrained** (*bool*) – If True, returns a model pre-trained on ImageNet
- **progress** (*bool*) – If True, displays a progress bar of the download to stderr

tensornet.models.**wide_resnet50_2**(*pretrained: bool = False*, *progress: bool = True*, ***kwargs: Any*) → tensornet.models.resnet.ResNet

Wide ResNet-50-2 model from "Wide Residual Networks". The model is the same as ResNet except for the bottleneck number of channels which is twice larger in every block. The number of channels in outer 1x1 convolutions is the same, e.g. last block in ResNet-50 has 2048-512-2048 channels, and in Wide ResNet-50-2 has 2048-1024-2048.

Parameters

- **pretrained** (*bool*) – If True, returns a model pre-trained on ImageNet
- **progress** (*bool*) – If True, displays a progress bar of the download to stderr

tensornet.models.**wide_resnet101_2**(*pretrained: bool = False*, *progress: bool = True*, ***kwargs: Any*) → tensornet.models.resnet.ResNet

Wide ResNet-101-2 model from "Wide Residual Networks". The model is the same as ResNet except for the bottleneck number of channels which is twice larger in every block. The number of channels in outer 1x1 convolutions is the same, e.g. last block in ResNet-50 has 2048-512-2048 channels, and in Wide ResNet-50-2 has 2048-1024-2048.

Parameters

- **pretrained** (*bool*) – If True, returns a model pre-trained on ImageNet

- **progress** (*bool*) – If True, displays a progress bar of the download to stderr

1.2.3 MobilenetV2

class tensornet.models.**MobileNetV2**(*num_classes=1000*, *width_mult=1.0*, *inverted_residual_setting=None*, *round_nearest=8*, *block=None*, *norm_layer=None*)

MobileNet V2

Note: This model inherits the `BaseModel` class.

Parameters

- **num_classes** (int, optional) – Number of classes. (default: 1000)
- **width_mult** (float, optional) – Width multiplier - adjusts number of channels in each layer by this amount. (default: 1.0)
- **inverted_residual_setting** (*optional*) – Network structure.
- **round_nearest** (int, optional) – Round the number of channels in each layer to be a multiple of this number. Set to 1 to turn off rounding. (default: 8)
- **block** (*optional*) – Module specifying inverted residual building block for mobilenet.
- **norm_layer** (*optional*) – Module specifying the normalization layer to use.

forward(*x*)

This function defines the forward pass of the model.

Parameters **x** (*torch.Tensor*) – Input.

Returns Model output.

Return type (*torch.Tensor*)

training: bool

tensornet.models.**mobilenet_v2**(*pretrained=False*, *progress=True*, ***kwargs*)

Constructs a MobileNetV2 architecture from "MobileNetV2: Inverted Residuals and Linear Bottlenecks".

Parameters

- **pretrained** (*bool, optional*) – If True, returns a model pre-trained on ImageNet. (default=False)
- **progress** (*bool, optional*) – If True, displays a progress bar of the download to stderr. (default=True)

1.2.4 Depth Estimation and Segmentation ResNet

class tensornet.models.**DSResNet**

A U-Net Inspired model for Monocular Depth Estimation and Image Segmentation.

For information check the Depth-Estimation-Segmentation repository.

Note: This model inherits the `BaseModel` class.

forward(*x: torch.Tensor*) → torch.Tensor

This function defines the forward pass of the model.

Parameters **x** (*torch.Tensor*) – Input.

Returns Model output.

Return type (*torch.Tensor*)

training: bool

1.3 Criterion

This module contains loss functions.

tensornet.models.loss.**cross_entropy_loss**()
Cross Entropy Loss. The loss automatically applies the softmax activation function on the prediction input.

Returns Cross entroy loss function

tensornet.models.loss.**bce_loss**()
Binary Cross Entropy Loss. The loss automatically applies the sigmoid activation function on the prediction input.

Returns Binary cross entropy loss function

tensornet.models.loss.**mse_loss**()
Mean Squared Error Loss.

Returns Mean squared error loss function

tensornet.models.loss.**rmse_loss**(*smooth=1e-06*)
Root Mean Squared Error Loss.

Returns Root mean squared error loss function

tensornet.models.loss.**dice_loss**(*smooth=1*)
Dice Loss.

Parameters **smooth** (float, optional) – Smoothing value. A larger smooth value (also known as Laplace smooth, or Additive smooth) can be used to avoid overfitting. (default: 1)

Returns Dice loss function

tensornet.models.loss.**bce_dice_loss**(*smooth=1e-06*)
BCE Dice Loss.

Parameters **smooth** (float, optional) – Smoothing value.

Returns BCE Dice loss function

tensornet.models.loss.**ssim_loss**(*data_range=1.0*, *size_average=True*, *channel=1*)
SSIM Loss.

Parameters

- **data_range** (float or int, optional) – Value range of input images (usually 1.0 or 255). (default: 255)
- **size_average** (bool, optional) – If size_average=True, ssim of all images will be averaged as a scalar. (default: True)
- **channel** (int, optional) – input channels (default: 1)

Returns SSIM loss function

tensornet.models.loss.**ms_ssim_loss**(*data_range=1.0*, *size_average=True*, *channel=1*)
MS-SSIM Loss.

Parameters

- **data_range** (float or int, optional) – Value range of input images (usually 1.0 or 255). (default: 1.0)
- **size_average** (bool, optional) – If size_average=True, ssim of all images will be averaged as a scalar. (default: True)
- **channel** (int, optional) – input channels (default: 1)

Returns MS-SSIM loss function

1.4 Optimizers

This module contains optimizers.

tensornet.models.optimizer.**sgd**(*model: torch.nn.modules.module.Module*, *learning_rate: float = 0.01*, *momentum: int = 0*, *dampening: int = 0*, *l2_factor: float = 0.0*, *nesterov: bool = False*)

SGD optimizer.

Parameters

- **model** (*torch.nn.Module*) – Model Instance.
- **learning_rate** (float, optional) – Learning rate for the optimizer. (default: 0.01)
- **momentum** (float, optional) – Momentum factor. (default: 0)
- **dampening** (float, optional) – Dampening for momentum. (default: 0)
- **l2_factor** (float, optional) – Factor for L2 regularization. (default: 0)
- **nesterov** (bool, optional) – Enables nesterov momentum. (default: False)

Returns SGD optimizer.

tensornet.models.optimizer.**adam**(*model: torch.nn.modules.module.Module*, *learning_rate: float = 0.001*, *betas: Tuple[float] = 0.9, 0.999*, *eps: float = 1e-08*, *l2_factor: float = 0.0*, *amsgrad: bool = False*)

Adam optimizer.

Parameters

- **model** (*torch.nn.Module*) – Model Instance.
- **learning_rate** (float, optional) – Learning rate for the optimizer. (default: 0.001)
- **betas** (tuple, optional) – Coefficients used for computing running averages of gradient and its square. (default: (0.9, 0.999))
- **eps** (float, optional) – Term added to the denominator to improve numerical stability. (default: 1e-8)
- **l2_factor** (float, optional) – Factor for L2 regularization. (default: 0)
- **amsgrad** (bool, optional) – Whether to use the AMSGrad variant of this algorithm from the paper On the Convergence of Adam and Beyond. (default: False)

Returns Adam optimizer.

1.5 Engine

Classes and methods used to train and test models.

class tensornet.engine.**Learner**(*train_loader*, *optimizer*, *criterion*, *device='cpu'*, *epochs=1*, *l1_factor=0.0*, *val_loader=None*, *callbacks=None*, *metrics=None*, *activate_loss_logits=False*, *record_train=True*)

Model Trainer and Validator.

Parameters

- **train_loader** (*torch.utils.data.DataLoader*) – Training data loader.
- **optimizer** (*torch.optim*) – Optimizer for the model.
- **criterion** (*torch.nn*) – Loss Function.
- **device** (str or torch.device, optional) – Device where the data will be loaded. (default='cpu')
- **epochs** (int, optional) – Numbers of epochs/iterations to train the model for. (default: 1)
- **l1_factor** (float, optional) – L1 regularization factor. (default: 0)
- **val_loader** (torch.utils.data.DataLoader, optional) – Validation data loader.
- **callbacks** (list, optional) – List of callbacks to be used during training.
- **metrics** (list, optional) – List of names of the metrics for model evaluation.

  Note: If the model has multiple outputs, then this will be a nested list where each individual sub-list will specify the metrics which are to be used for evaluating each output respectively. In such cases, the model checkpoint will consider only the metric of the first output for saving checkpoints.
- **activate_loss_logits** (bool, optional) – If True, the logits will first pass through the *activate_logits* function before going to the criterion. (default: False)
- **record_train** (bool, optional) – If False, metrics will be calculated only during validation. (default: True)

set_model(*model*)

Assign model to learner.

Parameters **model** (*torch.nn.Module*) – Model Instance.

update_training_history(*loss*)

Update the training history.

Parameters **loss** (*float*) – Loss value.

reset_history()

Reset the training history

activate_logits(*logits*)

Apply activation function to the logits if needed. After this the logits will be sent for calculation of loss or evaluation metrics.

Parameters **logits** (*torch.Tensor*) – Model output

Returns activated logits

Return type (*torch.Tensor*)

calculate_criterion(*logits*, *targets*, *train=True*)
Calculate loss.

Parameters

- **logits** (*torch.Tensor*) – Prediction.
- **targets** (*torch.Tensor*) – Ground truth.
- **train** (bool, optional) – If True, loss is sent to the L1 regularization function. (default: True)

Returns loss value

Return type (*torch.Tensor*)

fetch_data(*data*)
Fetch data from loader and load it to GPU.

Parameters **data** (tuple or list) – List containing inputs and targets.

Returns inputs and targets loaded to GPU.

train_batch(*data*)
Train the model on a batch of data.

Parameters **data** (tuple or list) – Input and target data for the model.

Returns Batch loss.

Return type (*float*)

train_epoch(*verbose=True*)
Run an epoch of model training.

Parameters **verbose** (bool, optional) – Print logs. (default: True)

train_iterations(*verbose=True*)
Train model for the 'self.epochs' number of batches.

evaluate(*loader*, *verbose=True*, *log_message='Evaluation'*)
Evaluate the model on a custom data loader.

Parameters

- **loader** (*torch.utils.data.DataLoader*) – Data loader.
- **verbose** (bool, optional) – Print loss and metrics. (default: True)
- **log_message** (*str*) – Prefix for the logs which are printed at the end.

Returns loss and metric values

validate(*verbose=True*)
Validate an epoch of model training.

Parameters **verbose** (bool, optional) – Print validation loss and metrics. (default: True)

save_checkpoint(*epoch=None*)
Save model checkpoint.

Parameters **epoch** (int, optional) – Current epoch number.

write_summary(*epoch*, *train*)
Write training summary in tensorboard.

Parameters

- **epoch** (*int*) – Current epoch number.
- **train** (*bool*) – If True, summary will be written for model training else it will be written for model validation.

fit(*start_epoch=1*, *epochs=None*, *reset=True*, *verbose=True*)
Perform model training.

Parameters

- **start_epoch** (int, optional) – Start epoch for training. (default: 1)
- **epochs** (int, optional) – Numbers of epochs/iterations to train the model for. If no value is given, the original value given during initialization of learner will be used.
- **reset** (bool, optional) – Flag to indicate that training is starting from scratch. (default: True)
- **verbose** (bool, optional) – Print logs. (default: True)

class tensornet.engine.**LRFinder**(*model*, *optimizer*, *criterion*, *metric='loss'*, *device=None*, *memory_cache=True*, *cache_dir=None*)
Learning rate range test. The learning rate range test increases the learning rate in a pre-training run between two boundaries in a linear or exponential manner. It provides valuable information on how well the network can be trained over a range of learning rates and what is the optimal learning rate.

Parameters

- **model** (*torch.nn.Module*) – Model Instance.
- **optimizer** (*torch.optim*) – Optimizer where the defined learning is assumed to be the lower boundary of the range test.
- **criterion** (*torch.nn*) – Loss function.
- **metric** (str, optional) – Metric to use for finding the best learning rate. Can be either 'loss' or 'accuracy'. (default: 'loss')
- **device** (str or torch.device, optional) – Device where the computation will take place. If None, uses the same device as *model*. (default: none)
- **memory_cache** (bool, optional) – If this flag is set to True, state_dict of model and optimizer will be cached in memory. Otherwise, they will be saved to files under the *cache_dir*. (default: True)
- **cache_dir** (str, optional) – Path for storing temporary files. If no path is specified, system-wide temporary directory is used. Notice that this parameter will be ignored if *memory_cache* is True. (default: None)

reset()
Restores the model and optimizer to their initial states.

range_test(*train_loader*, *iterations*, *mode='iteration'*, *learner=None*, *val_loader=None*, *start_lr=None*, *end_lr=10*, *step_mode='exp'*, *smooth_f=0.0*, *diverge_th=5*)
Performs the learning rate range test.

Parameters

- **train_loader** (*torch.utils.data.DataLoader*) – The training set data loader.
- **iterations** (*int*) – The number of iterations/epochs over which the test occurs. If 'mode' is set to 'iteration' then it will correspond to the number of iterations else if mode is set to 'epoch' then it will correspond to the number of epochs.

- **mode** (str, optional) – After which mode to update the learning rate. Can be either 'iteration' or 'epoch'. (default: 'iteration')
- **learner** (*Learner*, optional) – Learner object for the model. (default: None)
- **val_loader** (torch.utils.data.DataLoader, optional) – If None, the range test will only use the training metric. When given a data loader, the model is evaluated after each iteration on that dataset and the evaluation metric is used. Note that in this mode the test takes significantly longer but generally produces more precise results. (default: None)
- **start_lr** (float, optional) – The starting learning rate for the range test. If None, uses the learning rate from the optimizer. (default: None)
- **end_lr** (float, optional) – The maximum learning rate to test. (default: 10)
- **step_mode** (str, optional) – One of the available learning rate policies, linear or exponential ('linear', 'exp'). (default: 'exp')
- **smooth_f** (float, optional) – The metric smoothing factor within the [0, 1] interval. Disabled if set to 0, otherwise the metric is smoothed using exponential smoothing. (default: 0.0)
- **diverge_th** (int, optional) – The test is stopped when the metric surpasses the threshold: diverge_th * best_metric. To disable, set it to 0. (default: 5)

plot (*log_lr=True*, *show_lr=None*)

Plots the learning rate range test.

Parameters

- **skip_start** (int, optional) – Number of batches to trim from the start. (default: 10)
- **skip_end** (int, optional) – Number of batches to trim from the end. (default: 5)
- **log_lr** (bool, optional) – True to plot the learning rate in a logarithmic scale; otherwise, plotted in a linear scale. (default: True)
- **show_lr** (float, optional) – Is set, will add vertical line to visualize specified learning rate. (default: None)

1.6 Callbacks

1.6.1 Model Checkpoint

class tensornet.engine.ops.**ModelCheckpoint** (*path*, *monitor='val_loss'*, *mode='auto'*, *verbose=0*, *save_best_only=True*, *best_value=None*)

Store model checkpoint while training.

Parameters

- **path** (*str*) – Path to the directory where the checkpoints will be stored.
- **monitor** (str, optional) – Metric to monitor. (default: 'val_loss')
- **mode** (str, optional) – Comparison mode for monitored quantity. One of {auto, min, max}. (default: 'auto')
- **verbose** (int, optional) – verbosity mode, 0 or 1. (default: 0)

- **save_best_only** (bool, optional) – If True, only the model with the best value of monitoring quantity will be saved. (default: True)
- **best_value** (float, optional) – Best value of the monitored metric, this is useful when resuming training. This param will work only when *save_best_only* is True.

__call__(*model*, *current_value*, *epoch=None*, ***kwargs*)
Compare the current value with the best value and save the model accordingly.

Parameters

- **model** (*torch.nn.Module*) – Model Instance.
- **optimizer** (*torch.optim*) – Optimizer for the model.
- **current_value** (*float*) – Current value of the monitored quantity.
- **epoch** (*int*) – Epoch count.
- ****kwargs** – Other keyword arguments.

1.6.2 TensorBoard

class tensornet.engine.ops.**TensorBoard**(*logdir=None*, *images=None*, *device='cpu'*)
Setup Tensorboard.

Parameters

- **logdir** (str, optional) – Save directory location. Default is runs/CURRENT_DATETIME_HOSTNAME, which changes after each run.
- **images** (torch.Tensor, optional) – Batch of images for which predictions will be done.
- **device** (str or torch.device, optional) – Device where the data will be loaded. (default='cpu')

write_model(*model*)
Write graph to tensorboard.

Parameters **model** (*torch.nn.Module*) – Model Instance.

write_image(*image*, *image_name*)
Write image to tensorboard.

Parameters

- **image** (*torch.Tensor*) – Image tensor.
- **image_name** (*str, optional*) – Name of the image to be written.

write_images(*model*, *activation_fn=None*, *image_name=None*)
Write images to tensorboard.

Parameters

- **model** (*torch.nn.Module*) – Model Instance.
- **activation_fn** (*optional*) – Activation function to apply on model outputs.
- **image_name** (*str, optional*) – Name of the image to be written.

write_scalar(*scalar*, *value*, *step_value*)
Write scalar metrics to tensorboard.

Parameters

- **scalar** (*str*) – Data identifier.
- **value** (*float or string/blobname*) – Value to save.
- **step_value** (*int*) – Global step value to record.

1.6.3 LR Schedulers

tensornet.engine.ops.lr_scheduler.**step_lr**(*optimizer*, *step_size*, *gamma=0.1*, *last_epoch=-1*)

Create LR step scheduler.

Parameters

- **optimizer** (*torch.optim*) – Model optimizer.
- **step_size** (*int*) – Frequency for changing learning rate.
- **gamma** (float, optional) – Factor for changing learning rate. (default: 0.1)
- **last_epoch** (int, optional) – The index of last epoch. (default: -1)

Returns Learning rate scheduler.

Return type StepLR

tensornet.engine.ops.lr_scheduler.**reduce_lr_on_plateau**(*optimizer*, *factor=0.1*, *patience=10*, *verbose=False*, *min_lr=0*)

Create LR plateau reduction scheduler.

Parameters

- **optimizer** (*torch.optim*) – Model optimizer.
- **factor** (float, optional) – Factor by which the learning rate will be reduced. (default: 0.1)
- **patience** (int, optional) – Number of epoch with no improvement after which learning rate will be will be reduced. (default: 10)
- **verbose** (bool, optional) – If True, prints a message to stdout for each update. (default: False)
- **min_lr** (float, optional) – A scalar or a list of scalars. A lower bound on the learning rate of all param groups or each group respectively. (default: 0)

Returns ReduceLROnPlateau instance.

tensornet.engine.ops.lr_scheduler.**one_cycle_lr**(*optimizer*, *max_lr*, *epochs*, *steps_per_epoch*, *pct_start=0.5*, *div_factor=10.0*, *final_div_factor=10000*)

Create One Cycle Policy for Learning Rate.

Parameters

- **optimizer** (*torch.optim*) – Model optimizer.
- **max_lr** (*float*) – Upper learning rate boundary in the cycle.
- **epochs** (*int*) – The number of epochs to train for. This is used along with steps_per_epoch in order to infer the total number of steps in the cycle.

- **steps_per_epoch** (*int*) – The number of steps per epoch to train for. This is used along with epochs in order to infer the total number of steps in the cycle.
- **pct_start** (float, optional) – The percentage of the cycle (in number of steps) spent increasing the learning rate. (default: 0.5)
- **div_factor** (float, optional) – Determines the initial learning rate via initial_lr = max_lr / div_factor. (default: 10.0)
- **final_div_factor** (float, optional) – Determines the minimum learning rate via min_lr = initial_lr / final_div_factor. (default: 1e4)

Returns OneCycleLR instance.

tensornet.engine.ops.lr_scheduler.**cyclic_lr**(*optimizer*, *base_lr*, *max_lr*, *step_size_up=2000*, *step_size_down=None*, *mode='triangular'*, *gamma=1.0*, *scale_fn=None*, *scale_mode='cycle'*, *cycle_momentum=True*, *base_momentum=0.8*, *max_momentum=0.9*, *last_epoch=-1*, *verbose=False*)

Create Cyclic LR Policy.

Parameters

- **optimizer** (*torch.optim*) – Model optimizer.
- **base_lr** (*float*) – Lower learning rate boundary in the cycle.
- **max_lr** (*float*) – Upper learning rate boundary in the cycle.
- **step_size_up** (*int*) – Number of training iterations in the increasing half of a cycle. (default: 2000)
- **step_size_down** (*int*) – Number of training iterations in the decreasing half of a cycle. If step_size_down is None, it is set to step_size_up. (default: None)
- **mode** (*str*) – One of *triangular*, *triangular2*, *exp_range*. If scale_fn is not None, this argument is ignored. (default: 'triangular')
- **gamma** (*float*) – Constant in 'exp_range' scaling function: gamma**(cycle iterations). (default: 1.0)
- **scale_fn** – Custom scaling policy defined by a single argument lambda function, where 0 <= scale_fn(x) <= 1 for all x >= 0. If specified, then 'mode' is ignored. (default: None)
- **scale_mode** (*str*) – 'cycle', 'iterations'. Defines whether scale_fn is evaluated on cycle number or cycle iterations (training iterations since start of cycle). (default: 'cycle')
- **cycle_momentum** (*bool*) – If True, momentum is cycled inversely to learning rate between 'base_momentum' and 'max_momentum'. (default: True)
- **base_momentum** (*float*) – Lower momentum boundaries in the cycle. (default: 0.8)
- **max_momentum** (*float*) – Upper momentum boundaries in the cycle. Functionally, it defines the cycle amplitude (max_momentum - base_momentum). (default: 0.9)
- **last_epoch** (*int*) – The index of the last batch. This parameter is used when resuming a training job.(default: -1)
- **verbose** (*bool*) – If True, prints a message to stdout for each update. (default: False)

Returns CyclicLR instance.

1.7 Regularizers

tensornet.engine.ops.regularizer.**l1**(*model*, *loss*, *factor*)
Apply L1 regularization.

Parameters

- **model** (*torch.nn.Module*) – Model Instance.
- **loss** (*float*) – Loss function value.
- **factor** (*float*) – Factor for applying L1 regularization.

Returns Regularized loss value.

1.8 GradCAM

class tensornet.gradcam.**GradCAM**(*model: torch.nn.modules.module.Module*, *layer_name: str*)
Calculate GradCAM salinecy map.

Note: The current implemenation supports only ResNet models. The class can be extended to add support for other models.

Parameters

- **model** (*torch.nn.Module*) – A model instance.
- **layer_name** (*str*) – Name of the layer in model for which the map will be calculated.

saliency_map_size(**input_size*)
Returns the shape of the saliency map.

__call__(*input: tuple*, *class_idx: Optional[int] = None*, *retain_graph: bool = False*) → Tuple[torch.Tensor]

Parameters

- **input** (*tuple*) – Input image with shape of (1, 3, H, W)
- **class_idx** (int, optional) – Class index for calculating GradCAM. If not specified, the class index that makes the highest model prediction score will be used.

Returns

2-element tuple containing

- (*torch.tensor*): saliency map of the same spatial dimension with input.
- (*torch.tensor*): model output.

class tensornet.gradcam.**GradCAMPP**(*model: torch.nn.modules.module.Module*, *layer_name: str*)
Calculate GradCAM++ salinecy map.

It inherits the *GradCAM* class so the definition for all the methods is exactly the same as its parent class.

Note: The current implemenation supports only ResNet models. The class can be extended to add support for other models.

class tensornet.gradcam.**GradCAMView**(*model: torch.nn.modules.module.Module*, *layers: List[str]*, *device: Union[str, torch.device]*, *mean: Union[float, tuple]*, *std: Union[float, tuple]*)
Create GradCAM and GradCAM++.

Note: The current implemenation of *GradCAM* and *GradCAM++* supports only ResNet models. The class can be extended to add support for other models.

Parameters

- **model** (*torch.nn.Module*) – Trained model.
- **layers** (*list*) – List of layers to show GradCAM on.
- **device** (str or torch.device) – GPU or CPU.
- **mean** (float or tuple) – Mean of the dataset.
- **std** (float or tuple) – Standard Deviation of the dataset.

switch_mode()
Switch between GradCAM and GradCAM++.

cam(*norm_img_class_list: List[Union[Dict[str, Union[torch.Tensor, int]], torch.Tensor]]*)
Get CAM for a list of images.

Parameters **norm_img_class_list** (*list*) – List of dictionaries or list of images. If dict, each dict contains keys 'image' and 'class' having values 'normalized_image' and 'class_idx' respectively. class_idx is optional. If class_idx is not given then the model prediction will be used and the parameter should just be a list of images. Each image should be of type torch.Tensor

__call__(*norm_img_class_list: List[Union[Dict[str, Union[torch.Tensor, int]], torch.Tensor]]*) → List[Dict[str, Union[numpy.ndarray, Dict[str, numpy.ndarray]]]]
Get GradCAM for a list of images.

Parameters **norm_img_class_list** (*list*) – List of dictionaries or list of images. If dict, each dict contains keys 'image' and 'class' having values 'normalized_image' and 'class_idx' respectively. class_idx is optional. If class_idx is not given then the model prediction will be used and the parameter should just be a list of images. Each image should be of type torch.Tensor

tensornet.gradcam.**visualize_cam**(*mask: torch.Tensor*, *img: torch.Tensor*, *alpha: float = 1.0*) → Tuple[torch.Tensor]
Make heatmap from mask and synthesize GradCAM result image using heatmap and img.

Parameters

- **mask** (*torch.tensor*) – mask shape of (1, 1, H, W) and each element has value in range [0, 1]
- **img** (*torch.tensor*) – img shape of (1, 3, H, W) and each pixel value is in range [0, 1]

Returns

2-element tuple containing

- (*torch.tensor*): heatmap img shape of (3, H, W)
- (*torch.tensor*): synthesized GradCAM result of same shape with heatmap.

1.9 Utilities

This section lists out all the utility functions present througout `TensorNet`.

1.9.1 Common Utilities

`tensornet.utils.`**`set_seed`**(*seed: int*, *cuda: bool*)
Set seed to make the results reproducible.

Parameters

- **`seed`** (*`int`*) – Random seed value.
- **`cuda`** (*`bool`*) – Whether CUDA is available.

`tensornet.utils.`**`initialize_cuda`**(*seed: int*) → Tuple[bool, torch.device]
Check if GPU is availabe and set seed.

Parameters **`seed`** (*`int`*) – Random seed value.

Returns

2-element tuple containing

- (*bool*): if cuda is available
- (*torch.device*): device name

`tensornet.utils.`**`get_predictions`**(*model: torch.nn.modules.module.Module*, *loader: torch.utils.data.dataloader.DataLoader*, *device: Union[str, torch.device]*, *sample_count: int = 25*)
Get correct and incorrect model predictions.

Parameters

- **`model`** (*`torch.nn.Module`*) – Model Instance.
- **`loader`** (*`torch.utils.data.DataLoader`*) – Data Loader.
- **`device`** (`str` or `torch.device`) – Device where data will be loaded.
- **`(obj`** (*`sample_count`*) – *int*, optional): Total number of predictions to store from each correct and incorrect samples. (default: 25)

`tensornet.utils.`**`class_level_accuracy`**(*model: torch.nn.modules.module.Module*, *loader: torch.utils.data.dataloader.DataLoader*, *device: Union[str, torch.device]*, *classes: Union[List[str], Tuple[str]]*)
Print test accuracy for each class in dataset.

Parameters

- **`model`** (*`torch.nn.Module`*) – Model Instance.
- **`loader`** (*`torch.utils.data.DataLoader`*) – Data Loader.
- **`device`** (`str` or `torch.device`) – Device where data will be loaded.
- **`classes`** (`list` or `tuple`) – List of classes in the dataset.

`tensornet.utils.`**`plot_metric`**(*data: Union[List[float], Dict[str, List[float]]]*, *metric: str*, *title: str = None*, *size: Tuple[int] = 7, 5*, *legend_font: int = 15*, *legend_loc: str = 'lower right'*)
Plot accuracy graph or loss graph.

Parameters

- **data** (list or dict) – If only single plot then this is a list, else for multiple plots this is a dict with keys containing the plot name and values being a list of points to plot.
- **metric** (*str*) – Metric name which is to be plotted. Can be either loss or accuracy.
- **title** (str, optional) – Title of the plot, if no title given then it is determined from the x and y label.
- **size** (tuple, optional) – Size of the plot. (default: **'(7, 5)'**)
- **legend_loc** (str, optional) – Location of the legend box in the plot. No legend will be plotted if there is only a single plot. (default: *'lower right'*)
- **legend_font** (int, optional) – Font size of the legend (default: *'15'*)

tensornet.utils.**plot_predictions**(*data: List[dict]*, *classes: Union[List[str], Tuple[str]]*, *plot_title: str*, *plot_path: str*)

Display data.

Parameters

- **data** (*list*) – List of images, model predictions and ground truths. Images should be numpy arrays.
- **classes** (list or tuple) – List of classes in the dataset.
- **plot_title** (*str*) – Title for the plot.
- **plot_path** (*str*) – Complete path for saving the plot.

tensornet.utils.**save_and_show_result**(*classes: Union[List[str], Tuple[str]]*, *correct_pred: Optional[List[dict]] = None*, *incorrect_pred: Optional[List[dict]] = None*, *path: Optional[str] = None*)

Display network predictions.

Parameters

- **classes** (list or tuple) – List of classes in the dataset.
- **correct_pred** (list, optional) – Contains correct model predictions and labels.
- **incorrect_pred** (list, optional) – Contains incorrect model predictions and labels.
- **path** (str, optional) – Path where the results will be saved.

1.9.2 Model Utilities

Utility methods used by classes and methods present in the Models section.

tensornet.models.utils.**summary**(*model: torch.nn.modules.module.Module*, *input_size: Union[Tuple[int], List[int], Dict[str, Union[tuple, list]]]*, *batch_size: int = - 1*, *dtypes: Optional = None*)

Display model summary.

Parameters

- **model** (*torch.nn.Module*) – Model instance.
- **input_size** (tuple, list or dict) – Input size for the model.
- **batch_size** (int, optional) – Batch size. (default: -1)
- **dtypes** (*optional*) – Model input data types.

1.9.3 Data Utilities

Utility methods used by classes and methods present in the `Data` section.

`tensornet.data.utils.`**`unnormalize`**(*image*, *mean*, *std*, *transpose=False*)
Un-normalize a given image.

Parameters

- **`image`** (`numpy.ndarray` or `torch.Tensor`) – A ndarray or tensor. If tensor, it should be in CPU.
- **`mean`** (`float` or `tuple`) – Mean. It can be a single value or a tuple with 3 values (one for each channel).
- **`std`** (`float` or `tuple`) – Standard deviation. It can be a single value or a tuple with 3 values (one for each channel).
- **`transpose`** (`bool`, optional) – If True, transposed output will be returned. This param is effective only when image is a tensor. If tensor, the output will have channel number as the last dim. (default: False)

Returns Unnormalized image

Return type (*numpy.ndarray* or *torch.Tensor*)

`tensornet.data.utils.`**`normalize`**(*image*, *mean*, *std*, *transpose=False*)
Normalize a given image.

Parameters

- **`image`** (`numpy.ndarray` or `torch.Tensor`) – A ndarray or tensor. If tensor, it should be in CPU.
- **`mean`** (`float` or `tuple`) – Mean. It can be a single value or a tuple with 3 values (one for each channel).
- **`std`** (`float` or `tuple`) – Standard deviation. It can be a single value or a tuple with 3 values (one for each channel).
- **`transpose`** (`bool`, optional) – If True, transposed output will be returned. This param is effective only when image is a tensor. If tensor, the output will have channel number as the last dim. (default: False)

Returns Normalized image

Return type (*numpy.ndarray* or *torch.Tensor*)

`tensornet.data.utils.`**`to_numpy`**(*tensor*)
Convert 3-D torch tensor to a 3-D numpy array.

Parameters **`tensor`** (*`torch.Tensor`*) – Tensor to be converted.

Returns Image in numpy form.

Return type (*numpy.ndarray*)

`tensornet.data.utils.`**`to_tensor`**(*ndarray*)
Convert 3-D numpy array to 3-D torch tensor.

Parameters **`ndarray`** (*`numpy.ndarray`*) – Array to be converted.

Returns Image in tensor form.

Return type (*torch.Tensor*)

CHAPTER

TWO

CONTACT/GETTING HELP

If you need any help or want to report a bug, raise an issue in the repo.

PYTHON MODULE INDEX

t

INDEX

Symbols

A

B

C

D

E

F

G

I

L

M

N

O

P

R

S

T

U

V

W