What is tinyML?

tinyml is a small framework for writing neural networks for educational purposes. It is written in pure Python using third party libraries like numpy, tqdm etc..

How to install tinyML?

```
pip install tinyml
```

How it works?

- construct the neural networks
- perform the training, evaluating
- visualizing processes and export the trained weight to the persistent storage (i.e. the hard disk).

> We have core, Layer, losses, Net, Optimizer, learner

> Latest version of TinyML

Writing the Code: TinyML code leverages specialized frameworks like TensorFlow Lite Micro and Edge Impulse that optimize models for resource-constrained hardware. These frameworks typically use neural networks with shallower layers and smaller weights, ensuring efficient execution on tiny CPUs and MCUs.

Training the Model: Typically, training happens on powerful computers with plenty of data and processing power. You can choose open-source datasets like MNIST or collect your own sensor data. Frameworks often offer tools for data visualization, preprocessing, and model training, guiding you through the process.

Deployment on Hardware: Once trained, the model needs to be converted to a format compatible with the target hardware. Frameworks

provide tools for quantization and optimization, reducing the model size and computational demands. The code itself might involve running inference functions on the incoming sensor data and making decisions based on the model's predictions.

On the Edge: Deployed on tiny devices, the magic happens! Sensor data gets fed into the optimized model, generating predictions in real-time. This could be as simple as classifying an image on a camera, detecting anomalies in environmental readings, or optimizing energy consumption in a smart home. TinyML doesn't need internet connectivity, making it ideal for remote and low-power applications.

Training a basic CNN based neural network for handwritten digit recognition.

Import statements

```
import os

from sklearn.preprocessing import OneHotEncoder

import tinyml
import tinyml.dataloaders.mnist as mnist
from tinyml.core import Backend as np
from tinyml.layers import Conv2D, Dropout, Linear, ReLu,
Softmax, softmax
from tinyml.layers.flatten import Flatten
from tinyml.layers.pooling import MaxPool2D
from tinyml.learner import Learner
from tinyml.learner.callbacks import
evaluate_classification_accuracy
from tinyml.losses import cross_entropy_with_softmax_loss,
mse_loss
from tinyml.net import Sequential
from tinyml.optims import SGDOptimizer
```

Sets a verbose logging level, optionally enables GPU acceleration, and then signals the start of data loading.

```
# Higher verbose level = more detailed logging
tinyml.utilities.logger.VERBOSE = 1

GPU = False

if GPU:
    os.environ['TNN_GPU'] = "True"

print('loading data...')
# mnist.init()
```

Loads the MNIST dataset, Preprocesses the data, moves data to GPU

```
def pre_process_data(train_x, train_y, test_x, test_y):
    # Normalize
    train_x = train_x / 255.
    test_x = test_x / 255.
    train_x = train_x.reshape(-1, 1, 28, 28)
    test_x = test_x.reshape(-1, 1, 28, 28)

    return train_x, train_y, test_x, test_y

x_train, y_train, x_test, y_test = mnist.load()
x_train, y_train, x_test, y_test = pre_process_data(x_train,
y_train, x_test, y_test)

if GPU:
    import cupy as cp
    x_train = cp.array(x_train)
    y_train = cp.array(y_train)
    x_test = cp.array(x_test)
    y_test = cp.array(y_test)
```

```
print(y_train.shape)
print(x_train.shape)
print('building model...')
```

Defining the model and building it

```
model = Sequential([
    Conv2D('conv_1', (1, 28, 28),
           n_filter=32,
           h_filter=3,
           w_filter=3,
           stride=1,
           padding=0),
    ReLu('relu_1'),
    Conv2D('conv_2', (32, 26, 26),
           n_filter=64,
           h_filter=3,
           w_filter=3,
           stride=1,
           padding=0),
    ReLu('relu_2'),
    MaxPool2D('maxpool_1', (64, 24, 24), size=(2, 2), stride=2),
    Dropout('drop_1', 0.25),
    Flatten('flat_1'),
    Linear('fc_1', 9216, 128),
    ReLu('relu_3'),
    Dropout('drop_2', 0.5),
    Linear('fc_2', 128, 10),
])
model.build_params()
```

Getting the model summary

```
model.summary()
callbacks = [evaluate_classification_accuracy]
cargs = (x_test, y_test)
```

```
learner = Learner(model, cross_entropy_with_softmax_loss,
SGDOptimizer(lr=0.2))
```

Fitting the model with data

```
print('starting training...')
learner.fit(x_train,
            y_train,
            epochs=5,
            batch_size=1024,
            callbacks=callbacks,
            callbacks_interval=1,
            cargs=cargs)

print('training completed!')
```