Usage

keras2c can be used from the command line with the following syntax:

```
python -m keras2c [-h] [-m] [-t] model_path function_name

A library for converting the forward pass (inference) part of a keras model to
    a C function

positional arguments:
  model_path          File path to saved keras .h5 model file
  function_name       What to name the resulting C function

optional arguments:
  -h, --help          show this help message and exit
  -m, --malloc        Use dynamic memory for large arrays. Weights will be
                      saved to .csv files that will be loaded at runtime
  -t , --num_tests    Number of tests to generate. Default is 10
```

It can also be used with a python environment in the following manner:

```
from keras2c import k2c
k2c(model, function_name, malloc=False, num_tests=10, verbose=True)
```

In this case, `model` can be either the path to a saved `.h5` model on disk, or an instance of `keras.models.Model`

Using either of these methods will produce 3 files: `<function_name>.c`, `<function_name>.h`, and `<function_name>_test_suite.c`.

`<function_name>.c` will contain a function named `<function_name>` that when called replicates the forward pass (inference) through the neural network. The file will also include `initialize` and `terminate` functions for allocating and deallocating memory (other functionality can be added by the user). Additionally, if the model contains "stateful" elements such as RNN layers that maintain state between calls, a `reset` function will be generated to reset these states to zero.

By default, all of the weights and other parameters from the model are allocated as stack variables in the generated C code and the variables are declared directly in the generated function. For very large models or on machines with limited stack size this may lead to errors.

In such cases, the `malloc` option can be set to true, in which case array variables will be allocated on the heap via calls to `malloc` in the `initialize` routine, and the values will be written to `.csv` files that are read in as part of the `initialize` routine.

Tensor inputs and outputs to the generated function should use the type `k2c_tensor` defined in `include/k2c_tensor_include.h`:

```
struct k2c_tensor
{
    /** Pointer to array of tensor values flattened in row major order. */
    float *array;

    /** Rank of the tensor (number of dimensions). */
    size_t ndim;

    /** Number of elements in the tensor. */
    size_t numel;

    /** Array, size of the tensor in each dimension. */
    size_t shape[K2C_MAX_NDIM];
};
```

`<function_name>.h` is the header file associated with the generated source file, containing function declarations. `<function_name>_test_suite.c` contains a `main` program to run sample inputs through the generated code to ensure that it produces the same outputs as the original python model.

To compile and run the tests, the C backend must be built first. This can be done by running `make` from within the `include` folder, to generate `libkeras2c.a`. The test suite (or other main program) can then be compiled with:

```
gcc -std=c99 -I./include/ -o <executable_name> <function_name>.c
<function_name>_test_suite.c -L./include/ -l:libkeras2c.a -lm
```