Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für Integrierte Systeme
Integrated Systems Laboratory

Department of Information Technology and Electrical Engineering

Machine Learning on Microcontrollers

227-0155-00G

Exercise 7

Knowledge Distillation for Image Classification on STM32U5

ETH Center for Project-Based Learning

Wednesday 29th October, 2025

1 Introduction

In the previous exercise, we created a project using *STM Cube IDE*, set up our environment, performed real-time inference on the MCU, and explored in-depth quantization and pruning methods—two crucial techniques for Machine Learning on Microcontrollers—and compared their performance results. In this exercise session, we will focus on **Knowledge Distillation (KD)**, another key model compression technique. Besides learning how to train models using KD, we will conduct live inference on our `B-U585I-IOT02A` board using TensorFlow Lite, similar to our previous session. Additionally, we will explore how to generate optimized code from our *.tflite* or *.h5* models for our board using STM Cube AI, and we will compare outputs across different deployment scenarios.

2 Notation

Student Task: Parts of the exercise that require you to complete a task will be explained in a shaded box like this.

Note: You find notes and remarks in boxes like this one.

3 Knowledge Distillation

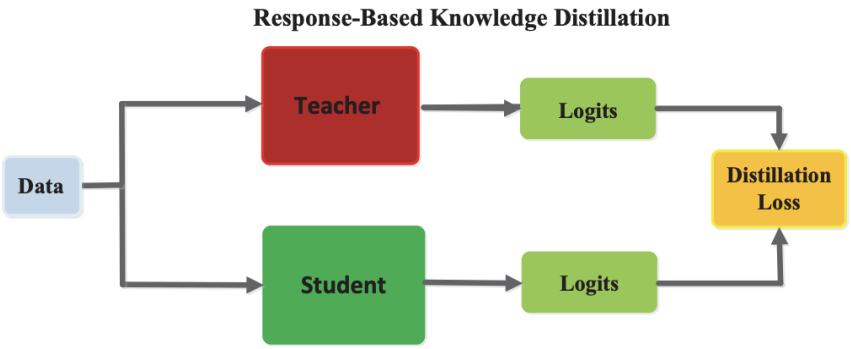

Figure 1: Response Based Knowledge Distillation

Knowledge Distillation (KD) is a powerful model compression technique used to transfer knowledge from a large, high-performing model (often referred to as the ”teacher” model) to a smaller, resource-efficient model (the ”student” model). This approach enables the smaller model to achieve higher performance than it would typically attain through direct training on the data alone.

The main concept behind KD involves leveraging the outputs of the teacher model, typically the soft probabilities (logits), as an additional training signal for the student model. By doing this, the student not only learns directly from the true labels but also captures the nuanced decision-making patterns of the teacher model, including subtle distinctions between classes that may not be evident from hard labels alone.

Practically, Knowledge Distillation is implemented by introducing a softening factor, called the "temperature," (T) which smooths the probability distribution output by the teacher.

$$p(z_i, T) = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)} \tag{1}$$

A higher temperature leads to softer probability distributions, making it easier for the student to learn nuanced relationships between classes. The student model is then trained using a combination of two loss functions: the traditional classification loss based on ground truth labels and a distillation loss based on the softened outputs of the teacher. The global loss is defined with (2).

$$\mathcal{L}_{\text{global}} = (1-\lambda)\,\mathcal{L}_{\text{CE}}\big(\psi(Z_s), y\big) \;+\; \lambda\,\tau^2\,\text{KL}\left(\psi\left(\frac{Z_s}{\tau}\right), \psi\left(\frac{Z_t}{\tau}\right)\right) \tag{2}$$

KD offers several advantages:

1. **Improved Model Efficiency**: Allows deployment of smaller models suitable for microcontrollers or edge devices with limited resources.
2. **Enhanced Generalization**: Helps the student model generalize better by providing rich information from the teacher's learned representations.
3. **Reduced Inference Time**: Smaller models result in faster inference, essential for real-time applications on constrained hardware platforms.

In this exercise, we will apply *Response Based Knowledge Distillation* to train optimized student models, deploy them on our STM32-based hardware platform, and evaluate their performance in real-time inference scenarios. Additionally, we will leverage STM Cube AI and TensorFlow Lite to generate optimized code and compare performance across different deployment strategies.

Note: In knowledge distillation, softmax outputs can be used instead of logits. In this exercise, the teacher and student models include a softmax activation in their final layers, enabling a direct comparison of their outputs. If you would like to use knowledge distillation in your projects, you could also experiment by training teacher models without a softmax in their final layer and compare the results accordingly.

4 Preparation

For this exercise, we will first train a model using the provided Jupyter Notebook code. Next, we will briefly train another model and compare it to a previously trained teacher model. Then, keeping the initial model's architecture unchanged, we will retrain our model using knowledge distillation with the teacher model. Since knowledge distillation leverages the teacher model's generalization capabilities to improve the student's generalization, it typically demonstrates greater effectiveness with longer training periods. Therefore, we provide you with the final version of the knowledge-distilled model as pre-trained. Finally, we will compare all models and perform live inference on our STM board using the best-performing model. Please make sure you have installed the environment in the first exercise on your system before you start.

Now, please activate your Docker environment and make your workspace ready.

5 Training a Convolutional Neural Network

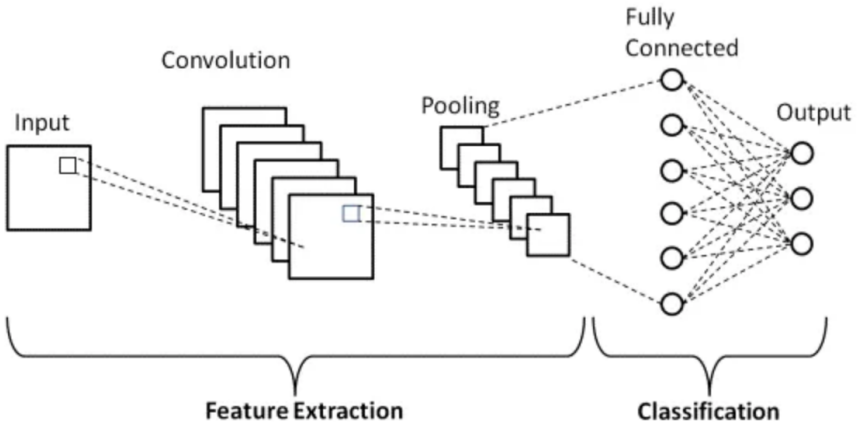

Figure 2: Basic CNN Architecture

A Convolutional Neural Network (CNN) is a type of deep learning model particularly suited for analyzing visual data such as images and videos. CNNs leverage spatial hierarchies through multiple layers of convolution operations, pooling, and fully connected layers to automatically detect meaningful patterns and features in data. The convolution layers apply filters to the input images, enabling the model to recognize features such as edges, textures, or shapes, while pooling layers help reduce spatial dimensions and computational load. CNNs have been widely used in various computer vision tasks, including image classification, object detection, and image segmentation, due to their ability to capture complex spatial structures efficiently.

For the first task, we will train a CNN model using the CIFAR-10 dataset. CIFAR-10 is a standard image classification dataset consisting of 60,000 colored images divided into 10 distinct classes (e.g., airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck). Each image has a resolution of 32x32 pixels. This dataset is widely used as a benchmark for training and evaluating computer vision algorithms and CNN models.

Student Task 1 (Training a CNN):

1. Open the provided *Jupyter Notebook* file and check if you have the correct version of TensorFlow.
2. Go to **Task 1** in the notebook and do all the steps provided.
3. How many parameters does the model have?

   Solution: 324394, it can be seen with the output of model.summary()
4. After the model training, what is the
   - Validation accuracy:

     Solution: $\sim$ 70.5%
   - Test accuracy:

     Solution: $\sim$ 70.72%

- Test loss:

  Solution: $\sim$ 81.52%

6 Quantizing the CNN

Last week, we conducted an in-depth analysis of quantization techniques. In this exercise, we will use one of the techniques examined last week: the Post-Training Integer Quantization method.

As a reminder, **Post-Training Quantization (PTQ)** is a technique used to optimize deep learning models by reducing their size and computational requirements while maintaining acceptable accuracy. PTQ is applied after training, converting model weights and activations from floating-point precision (e.g., FP32) to lower-bit representations such as INT8. Post-Training Integer Quantization specifically ensures that both weights and activations are fully quantized to integer values, making it highly efficient for deployment on hardware with limited computational resources, such as microcontrollers and edge devices. This technique significantly improves inference speed and reduces memory usage while enabling compatibility with specialized integer-based accelerators.

Student Task 2 (Quantization of the Model):

1. Go to **Task 2** in the notebook and do all the steps provided.
2. How did the accuracy of the model change after applying the post training quantization?

   Solution: It became $\sim$ 70.29%
3. Check the number of parameters in the model. Make a guess if the non-quantized model can fit on your microcontroller.

   Solution: It cannot fit because microcontroller has 2048 KiBinternal flash and 768 KiB RAM, while the model is 324394 * 4 bytes (FP32) = 1,297,576 bytes. This is greater than the available RAM (768 KiB) and would likely exceed the available flash when considering the program code. Therefore, it does not fit.
   Hint: Check your microcontroller's datasheet for its memory capacity.
4. Does the quantized network fit into your microcontroller? Calculate it.

   Solution: Yes, it should fit because it is 324394 Bytes after int8 quantization, which is approximately 317 KiB. This is well within the memory limits of the MCU.

7 Interpretation of the Teacher Model

In knowledge distillation, a *teacher model* is a larger and more well-trained model used to guide the *student model* (the smaller one). The teacher model can have a different architecture, be trained using different methods, and is generally a well-generalized model for the given dataset. Since the teacher model's generalization ability is transferred to the student, the student learns not only from the original ground truth labels but also from the teacher's predictions, allowing it to benefit from the teacher's learned representations.

However, the **teacher model is not a perfect model**—it has learned from the dataset and retains a certain level of error, meaning it does not always produce values that exactly match the ground truth

data. This also means that the teacher model transfers some of its errors to the student. Interestingly, when the teacher is a well-generalized model, **this imperfection can actually improve the student model's generalization**. By learning from the teacher's softened predictions rather than hard labels alone, the student can develop a more robust understanding of the dataset, often leading to better generalization and improved performance in real-world scenarios.

Because training the teacher model may last too long, in this exercise, we provide you with a *pre-trained teacher model* (teacher.h5), which has been trained on the CIFAR-10 dataset and achieves approximately 90% accuracy (The higher the teacher provides accuracy, the better the student model generalizes the data). This teacher model also follows a CNN architecture but is significantly larger and more complex than the student model. Its higher capacity allows it to generalize the dataset better, making it a strong reference for guiding the student model during knowledge distillation.

Student Task 3 (Interpretation of the Teacher Model):

1. Go to **Task 3** in the notebook and do all the steps provided.
2. How many parameters does the teacher model have?

   Solution: 3253834
3. Teacher model
   - Test accuracy:

     Solution: 88.97%
   - Test loss:

     Solution: 35.31%
4. How did the accuracy of the teacher model changed after quantization?

   Solution: It became 88.83%
5. Do you think the non-quantized or the quantized teacher model fits on your microcontroller?

   Solution: Both of them should not fit. The model has 3253834 parameters. The non-quantized one with FP32 requires 3253834 * 4 bytes (12710 KiB), while the quantized one needs 3253834 bytes (3178 KiB) with INT8 quantization. Neither will fit into the B-U585I-IOT02A microcontroller's 2048 KiB internal flash.
6. Visualize and compare the teacher model and small model. Follow the *.summary()* outputs as well.

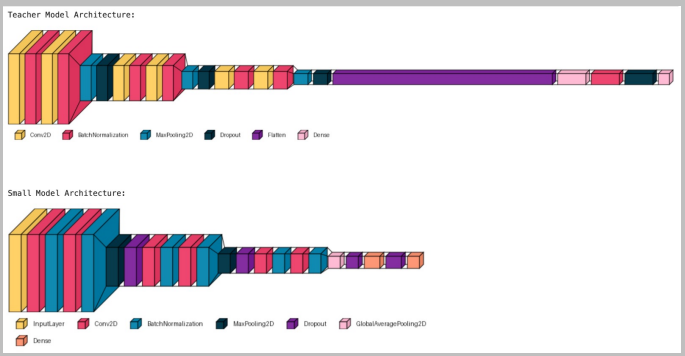

Figure 3: Model Visualization Output

8 Knowledge Distillation with the Teacher Model

So far, we have trained a small CNN network and compared its results with a previously trained larger model. Now, we will retrain our small network, but this time, we will redefine the loss function according to the distillation loss, as given in (2).

In *knowledge distillation*, two key parameters influence the process: **alpha** and **temperature**.

- Alpha (α) determines the balance between the student model's standard loss and the distillation loss. It controls how much the student model learns from the ground truth labels versus how much it learns from the teacher's predictions.
- Temperature (τ) affects how **softened** the outputs of the teacher and student models are. Higher temperature values produce softer probability distributions, allowing the student to capture more nuanced relationships between classes.

All these definitions and operations have been implemented in **Task 4** within the **Distiller class**.

Student Task 4 (Interpretation of the Teacher Model):

1. Go to **Task 4** in the notebook and read the explanation about **Distiller class**.
2. Try to match the mathematical definition of softmax with temperature (1) and distillation loss definition (2) in the Distiller class. And then, turn back to this sheet.

Now, we will train the model using knowledge distillation while also analyzing the impact of the selected parameters on the training process. By adjusting alpha (α) and temperature (T), we can observe how they influence the learning dynamics of the student model. Specifically, we will evaluate how different values of α affect the balance between direct supervision and distillation, and how varying T impacts the softness of the teacher's predictions. This analysis will help us understand the trade-offs involved in knowledge distillation and optimize the training process for better generalization.

Student Task 5 (Parameter Selection for Knowledge Distillation):

1. Set (*kd_alpha*, *kd_temperature*) as they are given below. First, please make a prediction about the parameters' effects, and then run the training to observe their effects. Compare especially losses of the training sessions (X is don't care).

   - (*kd_alpha*: X, *kd_temperature*: 0)

     Solution: When temperature is zero, accuracy of the model cannot improve because temperature is used as a division factor as it is given in (1). As a result, the model learns nothing.

```
Epoch 1/5
703/704 [============================>.] - ETA: 0s - accuracy: 0.1005 - distillation_loss: 0.0000e+00 - student_loss: nan
Epoch 1: val_accuracy improved from -inf to 0.09520, saving model to kd_checkpoint.h5
704/704 [==============================] - 91s 129ms/step - accuracy: 0.1005 - distillation_loss: 0.0000e+00 - student_loss: nan - val_accura
cy: 0.0952 - val_student_loss: nan
Epoch 2/5
703/704 [============================>.] - ETA: 0s - accuracy: 0.1006 - distillation_loss: 0.0000e+00 - student_loss: nan
Epoch 2: val_accuracy did not improve from 0.09520
704/704 [==============================] - 98s 139ms/step - accuracy: 0.1005 - distillation_loss: 0.0000e+00 - student_loss: nan - val_accura
cy: 0.0952 - val_student_loss: nan
Epoch 3/5
464/704 [==================>...........] - ETA: 29s - accuracy: 0.1000 - distillation_loss: 0.0000e+00 - student_loss: nan
```

Figure 4: Traing with temperature $= 0$

- (*kd_alpha*: 1, *kd_temperature*: 1)

  Solution: With alpha: 1, temperature: 1, knowledge distillation from teacher to student is not used. When we check (2) and Distiller class in the code, we can see that these hyperparameters make the student learn by itself. So, the output of training logs should be quite similar to the ones we have seen when we trained the model itself.

```
Epoch 1/5
703/704 [============================>.] - ETA: 0s - accuracy: 0.4548 - distillation_loss: 0.0467 - student_loss: 1.5148
Epoch 1: val_accuracy improved from -inf to 0.52420, saving model to kd_checkpoint.h5
704/704 [==============================] - 83s 118ms/step - accuracy: 0.4548 - distillation_loss: 0.0467 - student_loss: 1.5139 - val_accurac
y: 0.5242 - val_student_loss: 0.9196
Epoch 2/5
703/704 [============================>.] - ETA: 0s - accuracy: 0.5788 - distillation_loss: 0.0364 - student_loss: 1.1920
Epoch 2: val_accuracy improved from 0.52420 to 0.62200, saving model to kd_checkpoint.h5
704/704 [==============================] - 86s 123ms/step - accuracy: 0.5788 - distillation_loss: 0.0365 - student_loss: 1.1943 - val_accurac
y: 0.6220 - val_student_loss: 0.8961
Epoch 3/5
703/704 [============================>.] - ETA: 0s - accuracy: 0.6229 - distillation_loss: 0.0327 - student_loss: 1.0786
Epoch 3: val_accuracy improved from 0.62200 to 0.67100, saving model to kd_checkpoint.h5
704/704 [==============================] - 89s 126ms/step - accuracy: 0.6230 - distillation_loss: 0.0327 - student_loss: 1.0779 - val_accurac
y: 0.6710 - val_student_loss: 0.5851
Epoch 4/5
703/704 [============================>.] - ETA: 0s - accuracy: 0.6469 - distillation_loss: 0.0304 - student_loss: 1.0085
Epoch 4: val_accuracy improved from 0.67100 to 0.68560, saving model to kd_checkpoint.h5
704/704 [==============================] - 89s 127ms/step - accuracy: 0.6469 - distillation_loss: 0.0305 - student_loss: 1.0142 - val_accurac
y: 0.6856 - val_student_loss: 0.5126
Epoch 5/5
703/704 [============================>.] - ETA: 0s - accuracy: 0.6722 - distillation_loss: 0.0286 - student_loss: 0.9562
Epoch 5: val_accuracy improved from 0.68560 to 0.71120, saving model to kd_checkpoint.h5
704/704 [==============================] - 89s 127ms/step - accuracy: 0.6722 - distillation_loss: 0.0286 - student_loss: 0.9559 - val_accurac
y: 0.7112 - val_student_loss: 0.4091
<keras.src.callbacks.History at 0x359d379d0>
```

Figure 5: Traing with temperature $= 1$, alpha $= 1$

- (*kd_alpha*: 0, *kd_temperature*: 1)

  Solution: With alpha: 0, temperature: 1, knowledge distillation from teacher to student is fully used and the student does not make any contribution to its learning phase. The teacher takes the training inputs, makes an inference, and updates the weights of the student model. Even though the student does nothing for its own training, thanks to the teacher and distillation loss, it still learns the dataset. In this distillation, the student learns not only how to predict the input class but also learns about the other classes with only one inference from the teacher. This is the result of distillation of all the knowledge of the teacher to the student.

```
Epoch 1/5
703/704 [============================>.] - ETA: 0s - accuracy: 0.3939 - distillation_loss: 0.0514 - student_loss: 1.7664
Epoch 1: val_accuracy improved from -inf to 0.46640, saving model to kd_checkpoint.h5
704/704 [==============================] - 87s 124ms/step - accuracy: 0.3939 - distillation_loss: 0.0514 - student_loss: 1.7653 - val_accurac
y: 0.4664 - val_student_loss: 1.0030
Epoch 2/5
703/704 [============================>.] - ETA: 0s - accuracy: 0.5544 - distillation_loss: 0.0384 - student_loss: 1.3448
Epoch 2: val_accuracy improved from 0.46640 to 0.55760, saving model to kd_checkpoint.h5
704/704 [==============================] - 100s 142ms/step - accuracy: 0.5544 - distillation_loss: 0.0385 - student_loss: 1.3448 - val_accura
cy: 0.5576 - val_student_loss: 0.9854
Epoch 3/5
703/704 [============================>.] - ETA: 0s - accuracy: 0.5975 - distillation_loss: 0.0348 - student_loss: 1.2368
Epoch 3: val_accuracy improved from 0.55760 to 0.61420, saving model to kd_checkpoint.h5
704/704 [==============================] - 97s 138ms/step - accuracy: 0.5976 - distillation_loss: 0.0348 - student_loss: 1.2350 - val_accurac
y: 0.6142 - val_student_loss: 0.6968
Epoch 4/5
703/704 [============================>.] - ETA: 0s - accuracy: 0.6227 - distillation_loss: 0.0325 - student_loss: 1.1759
Epoch 4: val_accuracy did not improve from 0.61420
704/704 [==============================] - 99s 141ms/step - accuracy: 0.6227 - distillation_loss: 0.0325 - student_loss: 1.1787 - val_accurac
y: 0.5902 - val_student_loss: 0.7773
Epoch 5/5
703/704 [============================>.] - ETA: 0s - accuracy: 0.6445 - distillation_loss: 0.0305 - student_loss: 1.1120
Epoch 5: val_accuracy improved from 0.61420 to 0.65620, saving model to kd_checkpoint.h5
704/704 [==============================] - 101s 144ms/step - accuracy: 0.6445 - distillation_loss: 0.0305 - student_loss: 1.1112 - val_accura
cy: 0.6562 - val_student_loss: 0.3280
```

Figure 6: Traing with temperature $= 1$, alpha $= 0$

In this exercise, we have conducted an analysis using **Task 4** with different hyperparameter choices to keep the training time short while still observing the effects of knowledge distillation. As seen, even though the **student model's loss is not directly calculated from the ground truth labels**, it can still be trained using the **teacher's predictions**. However, finding the right student-teacher balance requires careful hyperparameter selection and longer training periods. Additionally, although knowledge distillation can be performed directly using the teacher model's logits, the teacher model used in this exercise includes a softmax layer at the end. This means that instead of using raw logits, the student learns from the **softened probability outputs** of the teacher.

Common values for alpha (α) range from 0.3 to 0.7, where lower values emphasize the distillation loss. A typical choice may be α = 0.5, which balances both.

For temperature (T), values typically range from 2 to 6, with higher values producing softer probability distributions, making it easier for the student to learn subtle relationships between classes.

To observe the impact of knowledge distillation on the test set, we provide you with a pre-trained model, *'knowledge_distilled_model_final.h5'*, which follows the **same architecture** as the model used in this exercise but has been trained for a much longer period to maximize distillation effectiveness, using α = 0.4 and T = 3.

Student Task 6 (Final Knowledge Distilled Model):

1. Go to **Task 5** in the notebook and do all the steps provided.
2. How many parameters does the provided student model have? Is it the same with the previous small model?

   Solution: Yes it is the same with the previous small model because this model has the same architecture. It has 324394 parameters.
3. Final model
   - Test accuracy:

     Solution: 86.94%
   - Test loss:

     Solution: 51.2%
4. How did the accuracy of the final model changed after quantization?

   Solution: It became 87.06%
5. With the inference time comparison, **student** model was

   Solution: $\sim$ 5.08x times faster than the **teacher** model.

9 Real-time Inference on Microcontroller with the Student Model

At the final stage, we conducted a teacher vs. student speed comparison, where results may vary depending on the computer used. However, this same comparison could not be performed on an MCU because the teacher model is too large to fit into the microcontroller's memory. Instead, the student model was specifically trained using knowledge distillation to be small enough to fit on the MCU while still achieving performance close to the teacher model.

At this stage, we will verify our trained model on the MCU and perform live inference using the provided code that communicates with the board via UART, *live_inference.py*. The codes takes your camera input and resizes it to 32x32 image before sending it to the MCU. After MCU runs inference on the camera frame, it reads the detected class and inference time that comes from MCU and prints it to the terminal. Now, plug in your MCU to your computer.

As we did in previous exercise sessions, follow these steps to set up the project for the B-U585I-IOT02A development board:

- Create a new project selecting your board. Make sure that target language is C.
- Reset the pinout configuration (Shortcut: CTRL + P).
- In **Middleware and Software Packages**, ensure that X-CUBE-AI is installed. If it is not installed on your system, download and install the latest version.
- Open the settings interface under **Software Packs** $\longrightarrow$ **X-CUBE-AI** and verify the Core option is enabled and the Application option is set to Validation.
- Go to Middlewares and Software Packages, click on X-CUBE-AI, and confirm that this module is activated.
- In the Add Network section, navigate to the directory containing the trained models and proceed with the following task.

Student Task 7 (Verification of the Model on the Microcontroller):

1. After selecting the proper model type (Keras or TFLite) try to **Analyze** all the models below. Try different **compression and optimization** selections while you are trying to analyze them for the microcontroller. Note your observations about the models.

   - Teacher Model (FP32)

     Solution: Does not fit the microcontroller with any combinations of optimizations
   - Teacher Model (INT8)

     Solution: Does not fit the microcontroller with any combinations of optimizations
   - Student Model (FP32)

     Solution: Does not fit the microcontroller with any combinations of optimizations
   - Student Model (INT8)

     Solution: Fits the microcontroller even with no additional optimization

2. Select the one that fits into your MCU and run **Validate on Target**. From the generated report, calculate the MACC/cycle and observe the inference time while the MCU runs @ 160 MHZ.

```
ST.AI Profiling results v1.2 - "cifar_model"
------------------------------------------------------------------------
nb sample(s)      :   10
duration          :   476.770 ms by sample (476.767/476.771/0.002)
macc              :   15363936
cycles/MACC       :   2.48
CPU cycles        :   [38,141,559]
used stack/heap   :   not monitored/0 bytes
------------------------------------------------------------------------

 Inference time per node
 ------------------------------------------------------------------------------------------------------------
 c_id    m_id   type                  dur (ms)       %    cumul  CPU cycles       name
 ------------------------------------------------------------------------------------------------------------
 0       0      Conv2dPool (0x109)      25.751    5.4%     5.4%  [  2,060,114 ]   ai_node_0
 1       1      EltwiseInt (0x114)      32.729    6.9%    12.3%  [  2,618,358 ]   ai_node_1
 2       2      EltwiseInt (0x114)      32.730    6.9%    19.1%  [  2,618,361 ]   ai_node_2
 3       3      Conv2D (0x103)         140.541   29.5%    48.6%  [ 11,243,319 ]   ai_node_3
 4       4      EltwiseInt (0x114)      28.514    6.0%    54.6%  [  2,281,105 ]   ai_node_4
 5       5      EltwiseInt (0x114)      28.514    6.0%    60.6%  [  2,281,114 ]   ai_node_5
 6       6      Pool (0x10b)             6.236    1.3%    61.9%  [    498,886 ]   ai_node_6
 7       7      Conv2D (0x103)          45.677    9.6%    71.5%  [  3,654,133 ]   ai_node_7
 8       8      EltwiseInt (0x114)      10.382    2.2%    73.6%  [    830,581 ]   ai_node_8
 9       9      EltwiseInt (0x114)      10.382    2.2%    75.8%  [    830,593 ]   ai_node_9
 10      10     Conv2D (0x103)          62.587   13.1%    88.9%  [  5,006,998 ]   ai_node_10
 11      11     EltwiseInt (0x114)       7.216    1.5%    90.5%  [    577,290 ]   ai_node_11
 12      12     EltwiseInt (0x114)       7.216    1.5%    92.0%  [    577,290 ]   ai_node_12
 13      13     Pool (0x10b)             1.585    0.3%    92.3%  [    126,792 ]   ai_node_13
 14      14     Conv2D (0x103)          18.817    3.9%    96.2%  [  1,505,365 ]   ai_node_14
 15      15     EltwiseInt (0x114)       1.308    0.3%    96.5%  [    104,666 ]   ai_node_15
 16      16     EltwiseInt (0x114)       1.308    0.3%    96.8%  [    104,668 ]   ai_node_16
 17      17     Conv2D (0x103)          13.807    2.9%    99.7%  [  1,104,596 ]   ai_node_17
 18      18     EltwiseInt (0x114)       0.057    0.0%    99.7%  [      4,557 ]   ai_node_18
 19      19     EltwiseInt (0x114)       0.033    0.0%    99.7%  [      2,608 ]   ai_node_19
 20      20     Pool (0x10b)             0.137    0.0%    99.7%  [     10,926 ]   ai_node_20
 21      21     Dense (0x104)            1.071    0.2%   100.0%  [     85,679 ]   ai_node_21
 22      22     Dense (0x104)            0.109    0.0%   100.0%  [      8,753 ]   ai_node_22
 23      23     NL (0x107)               0.060    0.0%   100.0%  [      4,807 ]   ai_node_23
 ------------------------------------------------------------------------------------------------------------
 total                                  476.770                  [ 38,141,559 ]
 ------------------------------------------------------------------------------------------------------------
```

Figure 7: Report of Validation on Target

Solution: In the report, CPU cycles was reported as 38,141,559 cycles. In the live inference, this # of cycles should be consistent

After analyzing and validating the model on the MCU, you are ready to run real-time inference on the microcontroller. To generate the code using the IDE, go to **Project Manager**, write your project name (if it is empty). Then, name your network as **cifar_model** (to be compatible with the provided main.c) in the part shown in 8. Hit *CTRL + S* or *Generate Code* button to generate your optimized code for the model.

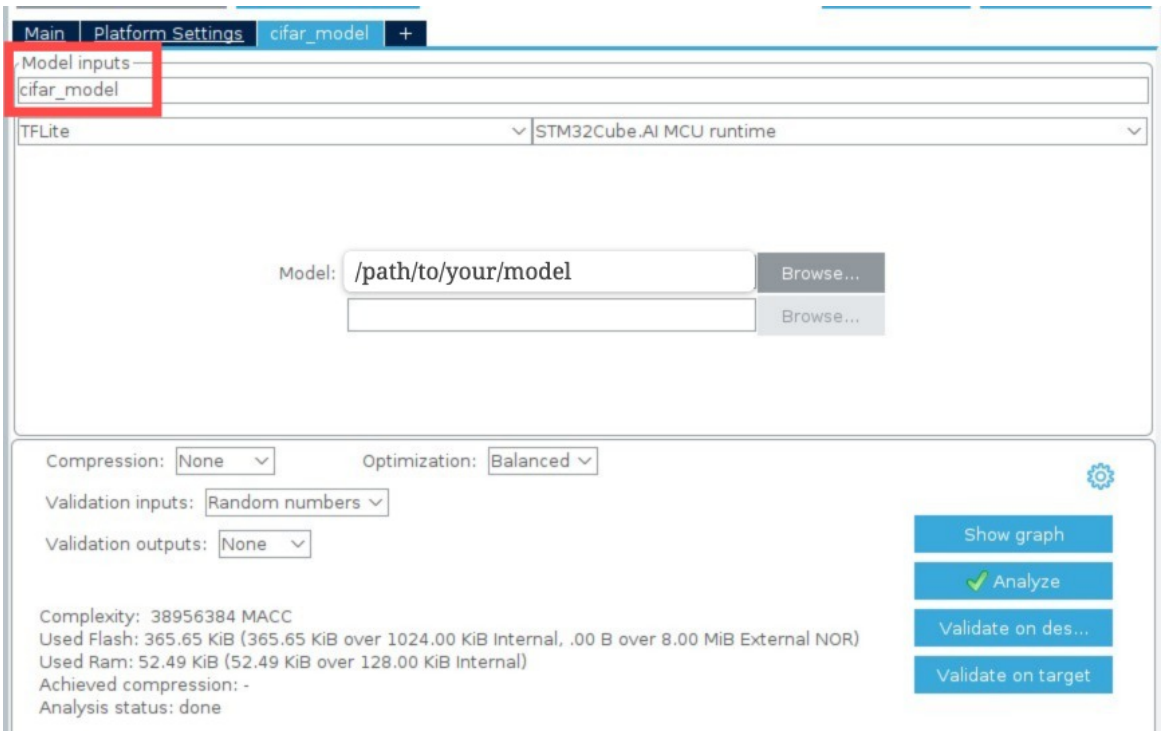

Figure 8: Generating the Code from the Project File

Before starting the task, check your inclusions are the same with 9, include directories may change depending on your OS and project folder, but you should see *Drivers, Middlewares, and X-CUBE-AI* in your included directories.

Figure 9: Project Include Directories

Note: After the code generation, you may face some build or compilation problems if your X-CUBE-AI version is not compatible with the given main.c code. Here, there are some possible issues that you may face and solutions for them:

- If you do not have an automatically generated **Middlewares** folder in your project, copy it from **provided_packages** folder that you have downloaded with this exercise.
- To disable X-CUBE-AI logs, go to *X-CUBE-AI/App/aiTestUtility.c* and find *void lc_print(const char* fmt, ...)* function and directly return it, or change the generated X-CUBE-AI with the provided one in **provided_packages** folder.
- If your IDE is generating seperate .c/.h files for your activated peripherals, you will see errors related to multiple definitions. Go to uart.c (if you have) and comment the lines related to **huart1 and USART1 initialization** since they are already defined in main.c.

If you face any other problem, you can ask for help.

Now, we have to fill the *main.c* file for real-time inference. Copy the provided main.c into the auto-generated one.

Student Task 8 (Real-time Inference on Microcontroller):

1. Go to the directory of your generated code and open the code. Open the generated *main.c* and replace it with the file that is given with this exercise. Familiarize yourself with the code. Fill the *TODO* part in the given main.c.
2. Check and learn the port that your computer communicates with your MCU. And then, run the *live_inference.py* code giving the communication port from the terminal, e.g python3 live_inference.py /dev/ttyUSB0. Now, your camera input is read with this Python code and sent to the MCU. Reset your MCU using its reset button, then you will see a message in your terminal "CIFAR10 Inference Demo is Started" after successful initialization. You may open images corresponding to CIFAR-10 classes using your phone and hold your phone close to your camera to test the model's inference performance.
3. What is the inference time on your MCU? Is it consistent with the STM generated report after the validation on target?

   Solution: In the report, CPU cycles was reported as 38,141,559 cycles. In the live inference, this # of cycles should be consistent, which corresponds to an inference time of approximately 238ms given the 160 MHz clock speed.

$\mathcal{E}$ **Congratulations! You have reached the end of the exercise.**
If you are unsure of your results, discuss them with an assistant. $\mathcal{E}$