Lab 4: Image Processing with CIFAR-10

Learning outcome: Develop a simple image classification model for CIFAR-10 using STM32CubeAI

Lab overview

In this lab, we will build a convolutional neural network for image classification. We will train the model with CIFAR-10 dataset, one of the most popular image datasets, which contains 60,000 images with 10 different categories. The model takes an RGB image and predicts the category of the image.

Requirements

Software functions

Python Packages

You will be using tensorflow, matplotlib, numpy, opencv-python, protobuf, and tqdm (version: 4.50.2). Install the required packages using Anaconda.

Main API Functions

Tensorflow

`tf.keras.datasets.cifar10.load_data()`
Load CIFAR-10 dataset

`tf.keras.Model.summary()`
Prints a string summary of the network

`tf.keras.Model.compile(optimizer, loss, metrics)`
Configures the model for training with the given optimizer, loss, and metrics

`tf.keras.Model.fit(x,y,batch_size,epochs,validation_data)`
Trains the model for a fixed number of epochs

Model Creation

Data preprocessing

First, open Jupyter Notebook through Anaconda Prompt.

```
> jupyter notebook
```

Open 'lab4.ipynb' on the notebook. Then, execute the first code block to import the required packages.

```
import tensorflow as tf

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Dropout, Flatten, MaxPooling2D,
BatchNormalization, Activation

import matplotlib.pyplot as plt
import matplotlib.image as mpimg

import numpy as np
import random

from PIL import Image

import os
```

Next, we are going to load CIFAR-10 dataset. We can easily get the dataset because TensorFlow provides API for downloading well-known datasets, such as CIFAR-10 and MNIST. Execute the next code block to get the dataset.

```
# Load data from TF Keras
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()

# CIFAR10 class names
class_names = ['Airplane', 'Automobile', 'Bird', 'Cat', 'Deer', 'Dog', 'Frog',
'Horse', 'Ship', 'Truck']
num_classes = len(class_names)
```

Then, we will save one image per class from the test set for testing with the board later. Execute the following code block to save the images.

```
path_images = "./Data/images/"

# Create directory
if not os.path.exists(path_images):
    os.mkdir(path_images)

# Save one image per class
ext=".jpg"
for image_index in range(0,100):
    im = Image.fromarray(x_test[image_index])
    im.save("./images/"+str(class_names[int(y_test[image_index])])+ext)
```

This code block will visualize the saved images.

```
# Show saved images
files = os.listdir(path_images)
for img in files:
    if os.path.splitext(img)[1] == ext and os.path.splitext(img)[0] in class_names:
        #print(os.path.splitext(img)[0])
        plt.subplot(2,5,class_names.index(os.path.splitext(img)[0])+1)
        plt.xticks([])
        plt.yticks([])
        plt.grid(False)
        plt.imshow(mpimg.imread(path_images+img),)
        plt.xlabel(os.path.splitext(img)[0])
plt.show()
```

Expected Output:

Next, we will normalize all the training and testing data to have values between 0 and 1. This normalization facilitates machine learning. Each RGB value ranges from 0 to 255, so we divide the training and testing data by 255.

```
# Normalize pixel values to be between 0 and 1
x_train = x_train.astype(np.float32)/255
x_test = x_test.astype(np.float32)/255

# Convert class vectors to binary class matrices.
y_train = tf.keras.utils.to_categorical(y_train, num_classes)
y_test = tf.keras.utils.to_categorical(y_test, num_classes)

# Print arrays shape
print('x_train shape:', x_train.shape)
print('y_train shape:', y_train.shape)
print('x_test shape:', x_test.shape)
print('y_test shape:', y_test.shape)
```

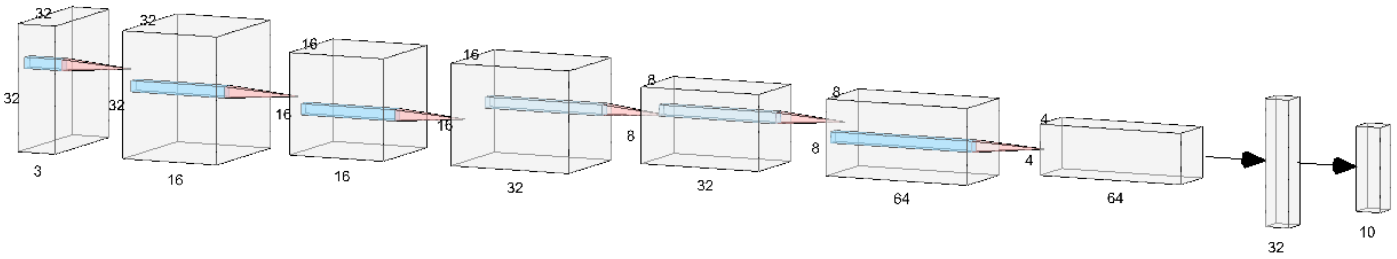
Expected Output:

```
x_train shape: (50000, 32, 32, 3)
y_train shape: (50000, 10)
x_test shape: (10000, 32, 32, 3)
y_test shape: (10000, 10)
```

Model creation

We are going to create a small convolutional neural network for image classification. The image size of CIFAR-10 is 32 by 32, and the number of colour channels is 3. So, the input shape of the first convolution layer is (32, 32, 3). Since the number of classes is 10, so the last dense layer should have 10 units.

Network architecture:



Note that only convolution and dense layers are illustrated in the figure.

```
# Hyperparameters
batch_size = 32
num_classes = len(class_names)
epochs = 1
img_rows, img_cols = x_train.shape[1], x_train.shape[2]
input_shape = (x_train.shape[1], x_train.shape[2], 1)
```

```
# Creating a Sequential Model and adding the layers
model = Sequential()
model.add(Conv2D(16, (3, 3), padding='same', input_shape=(32,32,3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Conv2D(16, (3, 3),padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2),strides=(2, 2)))
model.add(Dropout(0.2))

model.add(Conv2D(32, (3, 3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3),padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2),strides=(2, 2)))
```

```
model.add(Dropout(0.3))

model.add(Conv2D(64, (3, 3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3),padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2),strides=(2, 2)))
model.add(Dropout(0.4))

model.add(Flatten())
model.add(Dense(32))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(10)) #The number of classes we have
model.add(Activation('softmax'))
```

Execute the code blocks below to compile and train the model. If we use tens of epochs, the training could take more than 10 hours because the dataset has 50,000 training images. Therefore, the model trained for 50 epochs is provided for testing (File: 'Data/models/cifar10_model.h5'). You can use the model if you don't have enough time to train your own model.

```
# Check model structure and the number of parameters
model.summary()

# Let's train the model using Adam optimizer
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
```

```
# Train model
history = model.fit(x=x_train,
          y=y_train,
          batch_size=batch_size,
          epochs=epochs,
          validation_data=(x_test, y_test))
```

Let's save the model and evaluate the model. Note that since we trained the model for 1 epoch only, the accuracy would not be that good. Please try a larger number of epochs later to obtain better performance.

```
# Save keras model
path_models = "./Data/models/"
path_keras_model = path_models + "own_cifar10_model.h5"

# Create directory
if not os.path.exists(path_models):
    os.mkdir(path_models)
```

```
model.save(path_keras_model)

# Score trained model.
scores = model.evaluate(x_test, y_test, verbose=1)
print('Test loss:', scores[0])
print('Test accuracy:', scores[1])
```

Data Save

Finally, we are going to save the validation data and the labels for testing. This code block will sample 50 images from the dataset and save them in CSV format. Execute the code block to save the test data.

```
path_csv = "./Data/"
path_csv_file = path_csv+"own_cifar10_validation_20image.csv"

# Create directory
if not os.path.exists(path_csv):
    os.mkdir(path_csv)

# Remove old csv file
if os.path.exists(path_csv_file):
    os.remove(path_csv_file)

# Load data from TF Keras
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()


# CIFAR10 class names
class_names = ['Airplane', 'Automobile', 'Bird', 'Cat', 'Deer', 'Dog', 'Frog',
'Horse', 'Ship', 'Truck']

# Normalize pixel values to be between 0 and 1
x_train = x_train.astype(np.float32)/255
x_test = x_test.astype(np.float32)/255

# Print arrays shape
print('x_train shape:', x_train.shape)
print('y_train shape:', y_train.shape)
print('x_test shape:', x_test.shape)
print('y_test shape:', y_test.shape)

# Save csv file that contain pixel's value
num_sample = 50
rx = random.sample(range(0,len(x_test)),num_sample)

for i in range(0,num_sample):
    data = x_test[rx[i]]
    #print(data.shape)
```

```
    data = data.flatten()
    output = y_test[rx[i]]
    data=np.append(data,output)
    data = np.reshape(data, (1,data.shape[0]))
    #print(data.shape)
    with open(path_csv_file, 'ab') as f:
        np.savetxt(f, data, delimiter=",")
```

This code block will save the list of image classes. Execute the code block to save the label file.

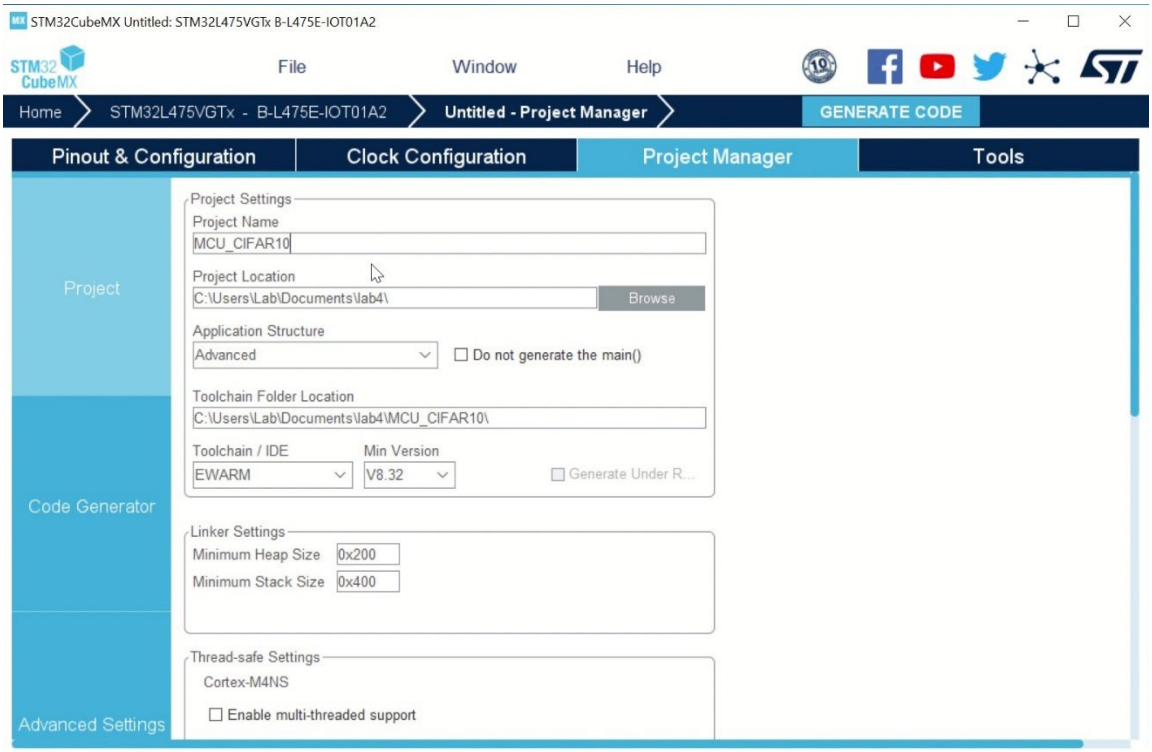
```
path_labels = "./Data/labels/”
path_labels_file = path_labels+"own_cifar10_labels.txt"

# Create directory
if not os.path.exists(path_labels):
    os.mkdir(path_labels)

# Remove old label file
if os.path.exists(path_labels_file):
    os.remove(path_labels_file)

# Create label file
for i in range(0,len(class_names)):
    with open(path_labels_file, 'a') as f:
        f.write(str(i)+","+class_names[i]+"\n")
```

Now we are ready to deploy the model on the evaluation board using STM32CubeAI. This code is largely contributed by Pau Danilo and Carra Alessandro from STMicroelectronics.

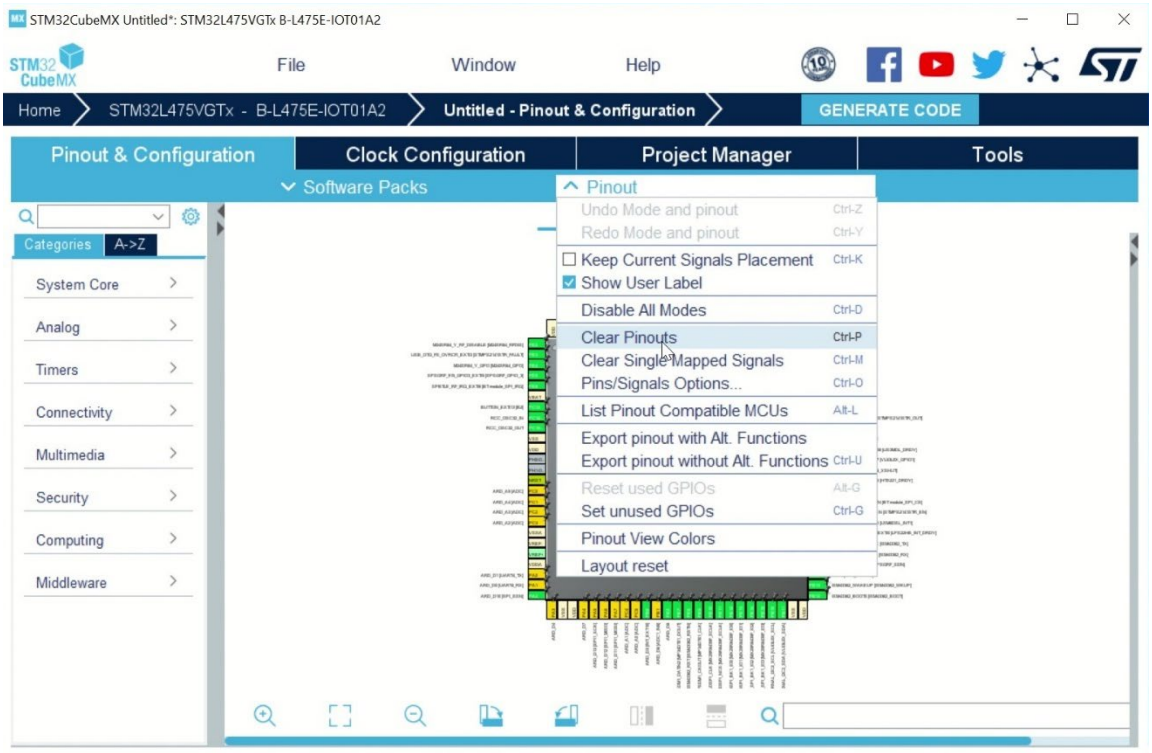
Deploying Model with STM32CubeMX

Model Deployment

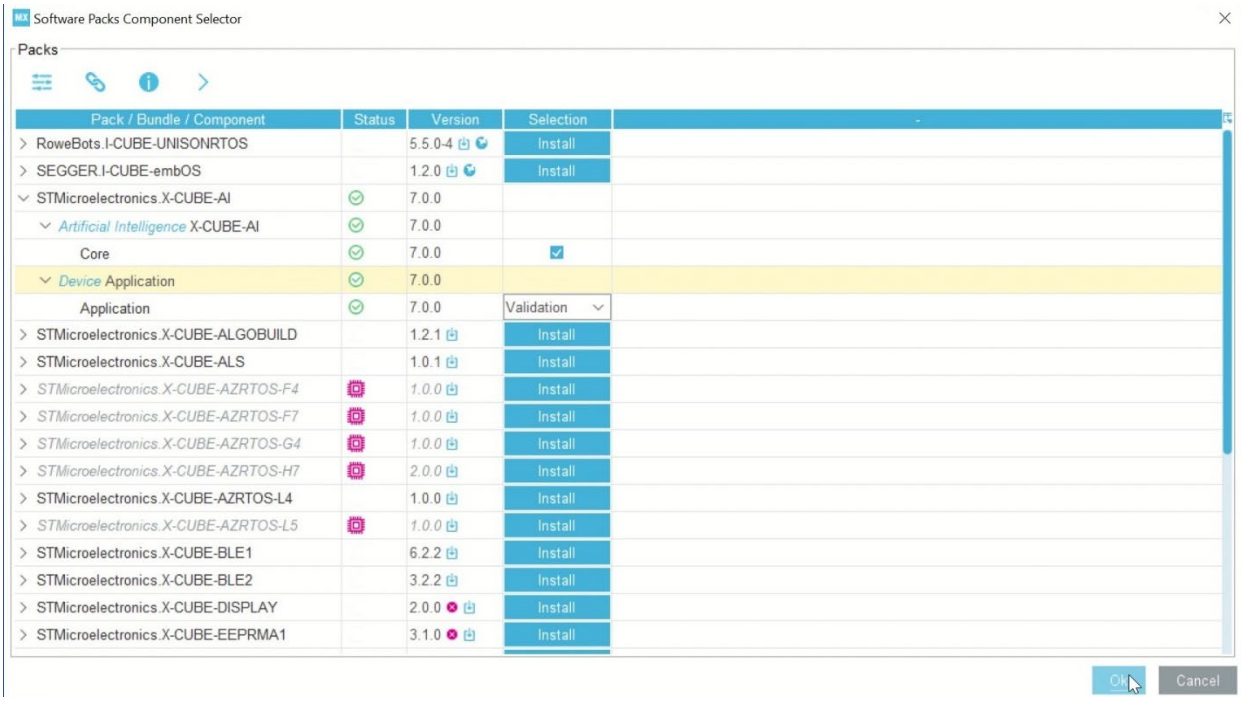
Open STM32CubeMX. Click [Access to Board Selector]. Find the B-L475E-IOT01A2 board and click [Start Project]. Go to [Project Manager]. Write the project name and configure the project location where the project will be saved. Select the firmware version to use.



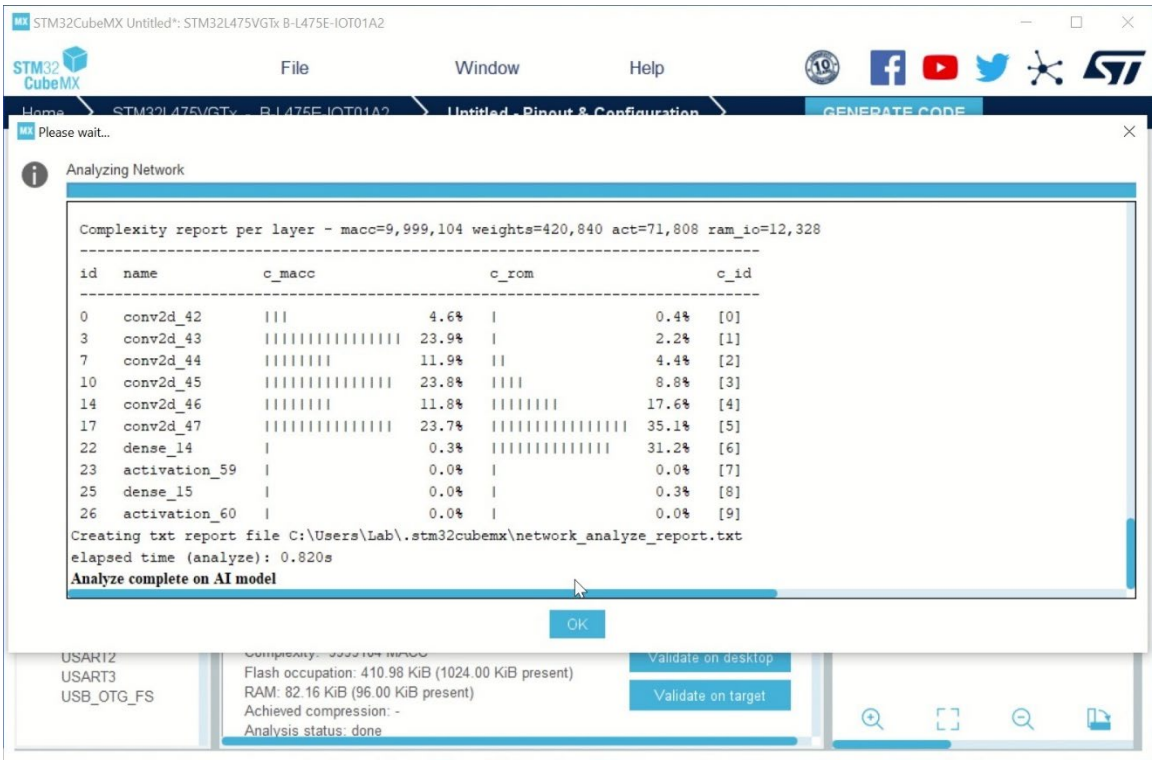
Go back to [Pinout & Configuration] and clear pinouts.



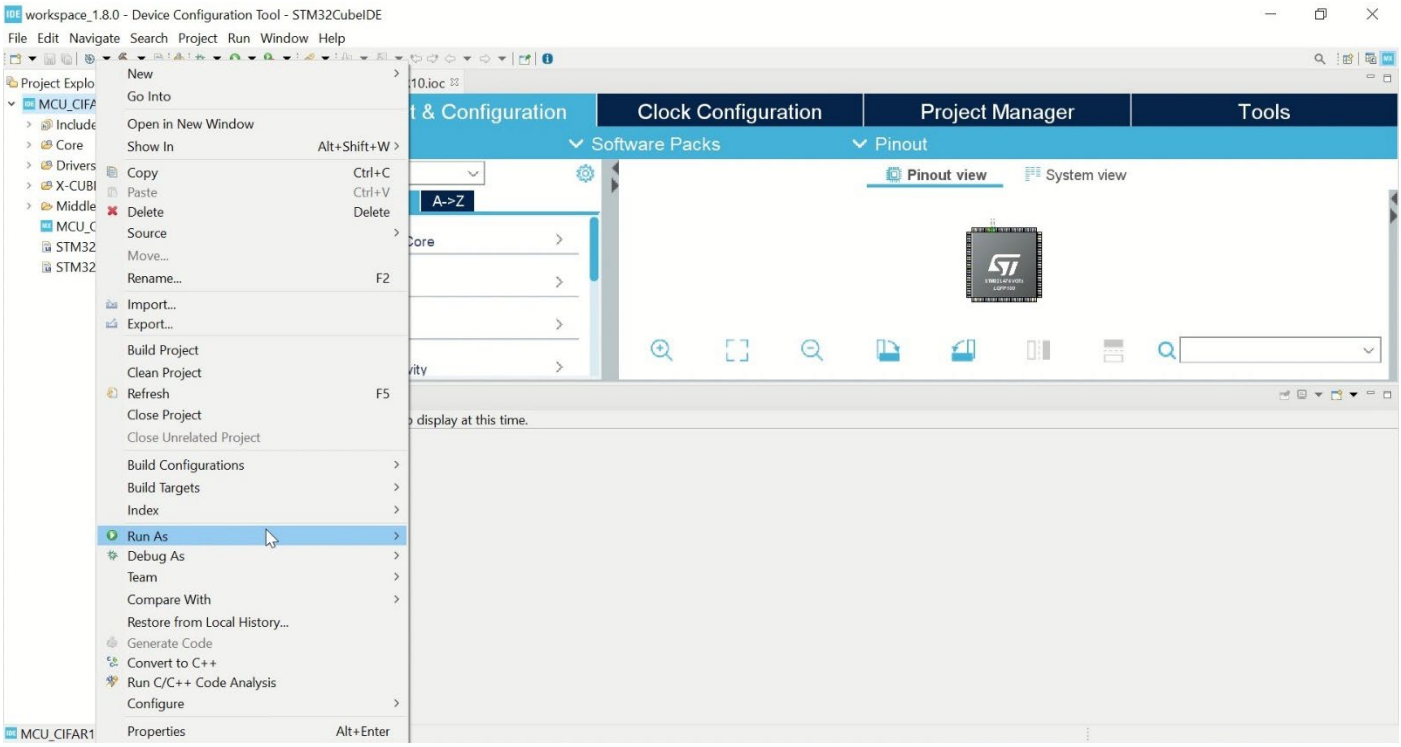
On [Software Packs] menu, click [Select Components]. Enable Cube AI. For device application, choose [Validation].



Click [Add network]. Load the model provided ('Data/models/cifar10_model.h5') or the model you trained by yourself ('Data/models/own_cifar10_model.h5'). Click [Analyze] to check the model. Then, generate the validation code for the model by clicking [Generate Code].



Open STM32CubeIDE. Choose 'Create a New STM32 Project from an Existing STM32CubeMX Configuration File' to import the project. Go to the project folder and open the .ioc file. Now, make sure that the board is connected to your computer. If it is correctly connected, install the code by clicking [Run As].



If you got the 'undefined reference' error, go to 'Core/Src/main.c'. In the file, remove static from the declaration of the 'MX_USART1_UART_Init' function and also from this definition. Then try [Run As] again.

```
static void MX_USART1_UART_Init(void);

static void MX_USART1_UART_Init(void)
{
```

Testing

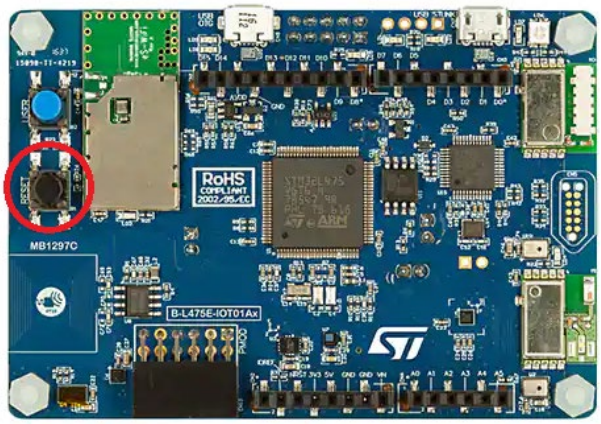
For testing, we are going to use a custom tool for sending images to the board. The tool is provided by STMicroelectronics, so all the copyright belongs to STMicroelectronics. Open Anaconda Prompt and activate your environment for the lab. We need to install several packages to run the tool.

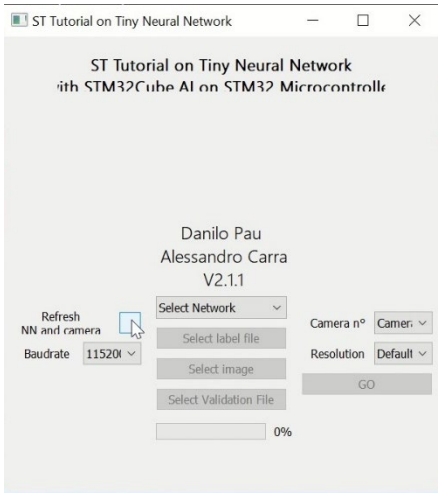
```
> python -m pip install -U opencv-python protobuf tqdm==4.50.2
```

Navigate to the lab4 folder. Enter [Misc] folder. Then you can execute the tool by typing the following command.

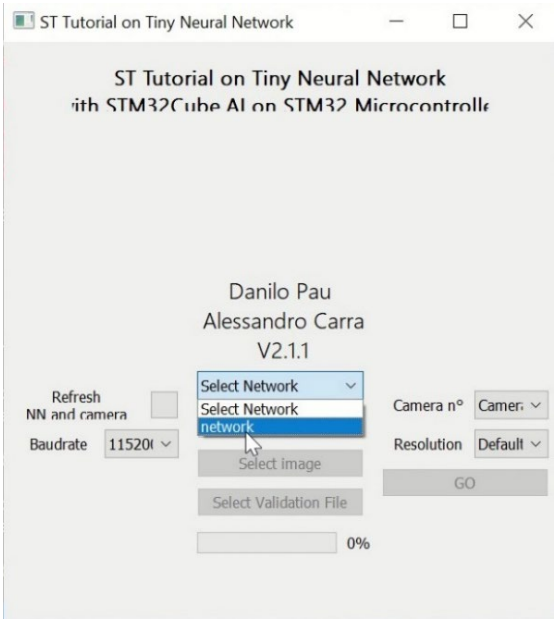
```
> cd ~/Documents/lab4/Misc
> python ui_python_ai_runner.py
```

Click the black button on the board to reset the board and check [Refresh NN and camera]. You can see the list of models deployed on the board.

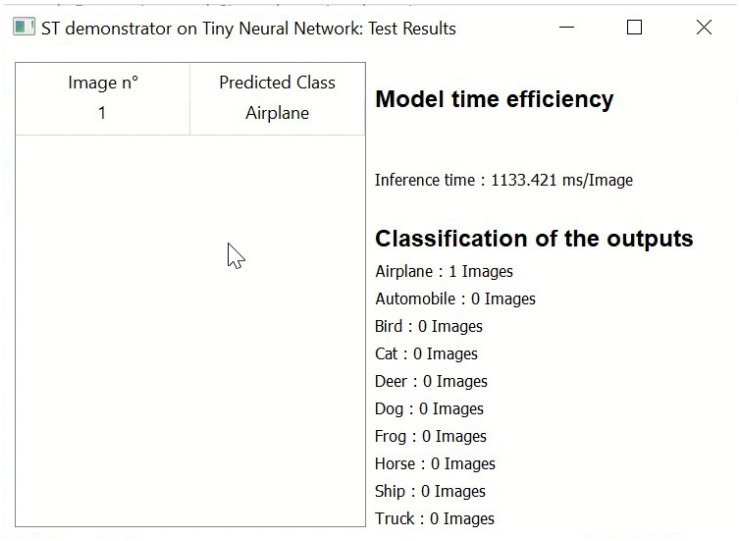




Select the network (‘Data/models/cifar10_model.h5’) and open the label file (‘Data/labels/cifar10_labels.txt’)



Open an image to test. The tool will automatically show the inference result. We can see that the model correctly predicted the label. In addition, we can see the time taken to finish the prediction.



You can also use your own camera to test image classification. If you press [S], the tool captures the image and sends it to the board.