AI:How to use transfer learning to perform image classification on STM32

Contents

Stable: 06.10.2021 - 15:40 / Revision: 06.10.2021 - 14:36

Stable: 25.05.2021 - 08:18 / Revision: 25.05.2021 - 08:17

The content format pdf is not supported by the content model wikitext.

Return to Main Page

Stable: 25.05.2021 - 08:18 / Revision: 25.05.2021 - 08:17

You do not have permission to edit this page, for the following reasons:

- The action you have requested is limited to users in one of the groups: Administrators, Editors, Reviewers, Selected_editors, ST_editors.
- The action "Read pages" for the draft version of this page is only available for the groups ST_editors, ST_readers, sysop, Selected_editors, Selected_readers, reviewer

---

You can view and copy the source of this page.

This article presents a video on how to use a technique called "Transfer learning" to quickly train a deep learning model in order to classify images. This video teaches you how to use ST ecosystem to build a computer vision application from the ground up. It describes how the FP-AI-VISION1 allows easy image collection with an STM32 Discovery kit, how to use transfer learning with Tensorflow to quickly train an image classification model, and eventually how to use STM32Cube.AI to convert this model into optimized code for ST microcontrollers. The Jupyter notebook used in this video is [https://github.com/STMicroelectronics/stm32ai/blob/master/AI_resources /VISION/transfer_learning/TransferLearning.ipynb available on Github]. It can be opened with Colab. [[File:Video ST CV TL.png|center||link=https://www.youtube.com/watch? v=NDshmSH7WnA&list=PLnMKNibPkDnG9IC5Nl9vJg1CKMAO1kODW|How to use transfer learning to perform image classification on STM32]] The article below describes step by step the video tutorial. For another complete example on how to use your own models with the FP-AI-VISION1 and Teachable Machine [[How to use Teachable Machine to create an image classification application on STM32|check out this article]]. In addition, [https://www.youtube.com/watch?v=ftM2D6qgrtI this series of videos] gives a good overview of the Vision function package. == Dataset generation == First you need a dataset of images representative of your classes. It can be generated either by following the steps described below or by using an already existing dataset (see related [https://github.com/STMicroelectronics/stm32ai/tree/master/AI_resources/VISION/transfer_learning Github page] for pasta datasets to jump start, note that it has been recorded with a fix distance of 13 cm and good light conditions). It is a good idea to use the same sensor for the dataset collection as the one that will be used for inference. In the present example, the [https://www.st.com/en/development-tools/b-cams-omv.html B-CAMS-OMV] board is used as camera module. Download the FP-AI-VISION1 function package from [https://www.st.com/en /development-tools/b-cams-omv.html https://www.st.com/en/embedded-software/fp-ai-vision1.html]. Using the USB Webcam application provided within FP-AI-VISION1, the board and the camera can be used as a webcam. First plug your board onto your computer using the '"ST-LINK"' port. Make sure the '"JP6"' jumper is set to '"ST-LINK"'. [[File:discordBoard.jpg|center|Discovery board ST-LINK setup]] After plugging the USB cable onto your computer, the board appears as a mounted device. The binary is located under <code>FP-AI-VISION1_V3.0.0\Projects\STM32H747I-DISCO\Applications\USB_Webcam\Binary</code>. Drag and drop it onto the board mounted device. This flashes the binary on the board. Unplug the board, change the '"JP6"' jumper to the '"HS"' position, and plug your board using the '"USB OTG"' port. [[File: discordBoard2.jpg|center|Discovery board USB OTG setup]] Launch the Windows® camera application (it can be found by searching for “camera” in the Windows search bar). If your computer has already a camera, change the source of the image feed by clicking the “flip camera” button. The capture can be done either in '"VGA"' or '"QVGA"', the resolution can be changed with the menu <code>settings -> Photos -> Quality</code>. The images will be resized to the input resolution of the neural network in the Python notebook. Take enough picture samples to fill your dataset. It must be structured as follows: <pre>Dataset cats cat0001.jpg cat0002.jpg ... dogs horses</pre> The dataset must to be large enough. The one used for the video includes 1408 files belonging to 8 classes. It is recommended to use at least 100 pictures per class. == Model generation == The Python notebook used to generate the model can be found at https://github.com /STMicroelectronics/stm32ai/blob/master/AI_resources/VISION/transfer_learning/TransferLearning.ipynb It can be open either with Google Colab (as in the present example) or by using any Python environment. Just click (or <code>Ctrl + Click</code>) the <code>Open in Colab</code> icon at the top of the notebook on Github. This takes you to the Colab notebook project. Zip your dataset folder and import it to the Colab project. [[File:ColabImport.png|center|Colab import]] You can run the script either step by step or run the whole script by pressing <code>Ctrl + F9</code>. All the steps are described in details in the script but what they

do is straightforward : * Import all the necessary libraries. * Unzip the dataset. * Import the dataset (this step also automatically resizes images to match the model input size). * Display samples from the dataset * Perform data augmentation by adding duplicate images with random contrast/translation/rotation/zoom. * Display new samples. * Normalize data. * Import a pretrained <code>MobileNet</code> model without the last layer and add new ones at the end. * Train the model. * Plot training loss history. * Quantize the model to reduce its size. * Export the quantized model. The trained and quantized model appears as <code>model_quant.tflite</code>. == Model deployment == This section describes how to generate a model that can be used on STM32 platforms, starting from a model produced by a high-level library such a Keras/Tensorlflow. It also describes how to use your model in an already existing project. === Model generation for STM32 === {{Warning|Warning, the latest version of FP-AI-VISION1 is based on STM32Cube.AI 6.0.0. Make sure your are using this version.}} First convert the <code>tflite</code> model to be used on a STM32 platform. There are two ways to achieve this goal, one by using STM32CubeMX and generating an entire project (recommended at first), the other by using the command line interface to generate only the required files. ==== Project generation with STM32CubeMX (Option 1) ==== Start STM32CubeMX and create a new project. In this example, the STM32H747I-DISCO board is used. Select

Start STM32CubeMX and create a new project. In this example, the STM32H747I-DISCO board is used. Select the board in the <code>Board Selector</code> and start the project. A new window opens showing the board configuration. Click <code>Software Packs -> Select Components</code>. [[File: CubeMXScreenShot.png|center|Adding components to STM32CubeMX]] Select the components for the Arm® Cortex®-M7 core under STMicroelectronics X-CUBE-AI, check the core box, select <code>Application</code> as application, then click Ok. [[File: CubeMXScreenShot2.png|center|Components selector]] The stm32-ai library will be download if it has not been downloaded yet. When it is complete, click Ok. Go to <code>Software Packs (on the left) -> STMicroelectronics X-CUBE AI</code>. A pop up window opens to configure peripherals and clock.Click Yes. Click <code>Add network</code>, call your network <code>network</code>, select TFLite from the scrollable menu, and browse to the last section to find the generated model. Once the model is selected, click <code>Analyze< /code>. This provides a quick overview of the model memory footprint. Click the <code>Show graph</code> button for more in-depth information on the model topology and RAM memory usage layer by layer including activation buffers. [[File:Graph1.png|center|Memory Usage windows]] To customize how the model is stored in memory, click the gear icon (<code>Advanced Settings</code>) and specify if you want to use external Flash memory and/or RAM. Check the <code>Use activation buffer for input buffer</code> to make them share the same buffer in order to reduce memory usage. [[File: CubeMXScreenShot3.png|950px|center|X-Cube-AI model loading]] Click <code>Analyze</code> again. You can see that the RAM occupation has been significantly reduced (refer to the memory graph for more details). [[File:Graph2.png|center|Memory Usage window after enabling input buffer space sharing]] < /div> Check that the model works properly by validating it either on the PC or on the target. To generate the code for STM32CubeIDE (or Arm® KEIL® or IAR SYSTEMS®) go to <code>Project manager</code>, specify a project name and location. Select STM32CubeIDE as toolchain, then click <code>GENERATE CODE</code> on the top right corner. [[File:CubeMXScreenShot4. png|950px|center|Project manager window]] This generates the necessary code for the whole project. In the present example, only the files related to the neural network are required. They are located under <code>YourProjectName\CM7\X-CUBE-AI\App\</code> and are called <code>network.c</code>,<code>network. h</code> ,<code>network_data.c</code>, <code>network_data.h</code> and <code>network_config.h</code>. They represent the model and its weight. ==== Command line interface (Option 2) ==== The other way to generate the necessary files is to use the command line interface. The main advantage is to produce only the required files. The documentation related to X CUBE AI is located under <code>C:\Users\<USERNAME> \STM32Cube\Repository\Packs\STMicroelectronics\X-CUBE-AI\6.0.0\Documentation/</code>. The CLI is called <code>stm32ai.exe</code> for Windows. It is located under <code>C:\Users\<USERNAME> \STM32Cube\Repository\Packs\STMicroelectronics\X-CUBE-AI\6.0.0\Utilities\windows</code> and <code>$HOME /STM32Cube/Repository/Packs/STMicroelectronics/X-CUBE-AI/6.0.0/Utilities/</code> for UNIX®-based systems. If there is no <code>utilities</code> folder, create a CUBE MX project and add Core AI in the Software Package (see section Project generation). This downloads all the necessary files. For ease of usage, add the folder to your Windows path: {{PC$}} set CUBE_FW_DIR=C:\Users\<USERNAME> \STM32Cube\Repository {{PC$}} set X_CUBE_AI_DIR=%CUBE_FW_DIR%\Packs\STMicroelectronics\X-CUBE-AI\6.0.0 {{PC$}} set PATH=%X_CUBE_AI_DIR%\Utilities\windows;%PATH% Once added, go to the folder where you wish to generate the network files in command line (using PowerShell or similar) and run the following command: {{PC$}} stm32ai.exe generate -m .\model_quant.tflite --allocate-inputs # (change the path to the model if needed). This creates all the necessary files under the <code>stm32ai_output</code> folder. === Model integration === Now that the C files are generated for your model, integrate them in an already existing project. In this example, the application is based on the '"Person detection" AI function package project. Launch the STM32CubeIDE project with the <code>.project</code> located under <code>FP-AI-VISION1_V3.0.0\FP-AI-VISION1_V3.0.0\Projects\STM32H747I-DISCO\Applications\PersonDetection\MobileNetv2_Model\STM32CubeIDE</code>. Double click it to open STM32CubeIDE. Select the <code>CM7</code> subproject and build it. The resulting workspace is shown below: [[File:CubeIDEScreenShot.png|center|STM32CubeIDE Project window]] Replace the model files (<code>network.c</code>,<code>network.h</code> ,<code>network_data.c</code>, <code>network_config.h</code> and <code>network_config.h</code>) by those generated previously with STM32CubeMX. Replace them in Windows explorer. Below the file location: <pre>PersonDetection +-- MobileNetv2_Model | +-- Inc | | network.h | | network_data.h | | network_config.h | +-- Src | | network.c | | network_data.c</pre> Change the name of your classes. Go to <code>fp_vision_app.c</code> and change <code>output_labels</code> to your class names. Be careful that there are as many labels as your model output size. Clean your project under STM32CubeIDE with <code>Project->Clean...</code>. Make sure that your board is connected to the ST-LINK port and properly powered (set the jumper position to <code>ST-LINK<

board is connected to the ST-LINK port and properly powered (set the jumper position to <code>ST-LINK</ /code>). Build your project and run it with the play button. A ST-Application pack splash window appears followed by the camera view and the model classification. If either of these window is not displayed properly, go through the wiki article to check if steps have been skipped. <noinclude> [[Category:Artifical Intelligence|11]] {{PublicationRequestId | 20512| 2021-07-12 | AnneJ ?}} </noinclude>

Template used on this page:

- Template:PublicationRequestId (view source)

Return to Main Page.