L03_Eduardo_Cabrera-Lopez_ITAI3377 - Simulation Documentation

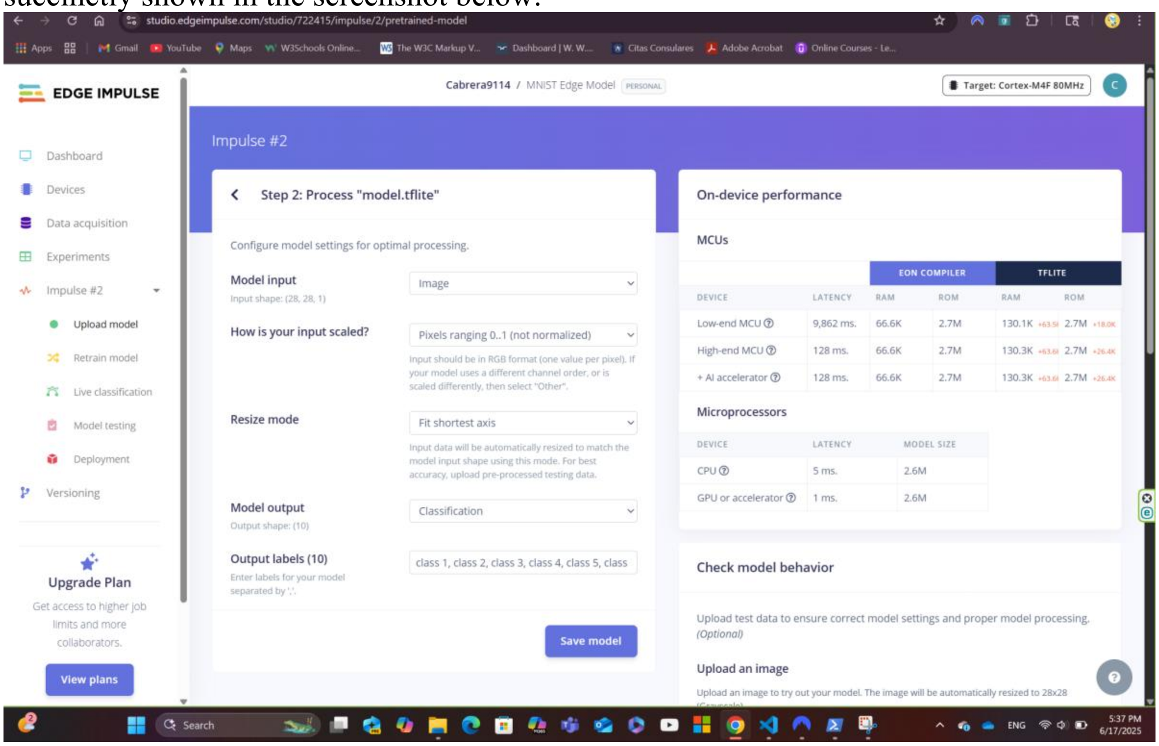
Edge Impulse Model Upload

Following TensorFlow Lite conversion of the trained model, I uploaded the model.tflite file to Edge Impulse platform via online dashboard. This stage was required since the CLI, edge-impulse-uploader, only enables uploading datasets including photos, audio, and sensor data—not model files.

Edge Impulse immediately examined the model and reported its performance traits over several hardware profiles once the file was uploaded under the Deployment > Custom Models section. For systems like the Cortex-M4F, Cortex-M7, and Raspberry Pi 4, this covered measures including RAM use, ROM size, and inference delay.

These revelations help one decide whether a model fits for use on limited edge devices. In this scenario, the model displayed low latency and memory use and satisfied the criteria for multiple hardware configurations. This verified the success of the conversion procedure and the good optimization of the model for edge situations.

The performance of the uploaded model on Edge Impulse's evaluation dashboard is succinctly shown in the screenshot below.



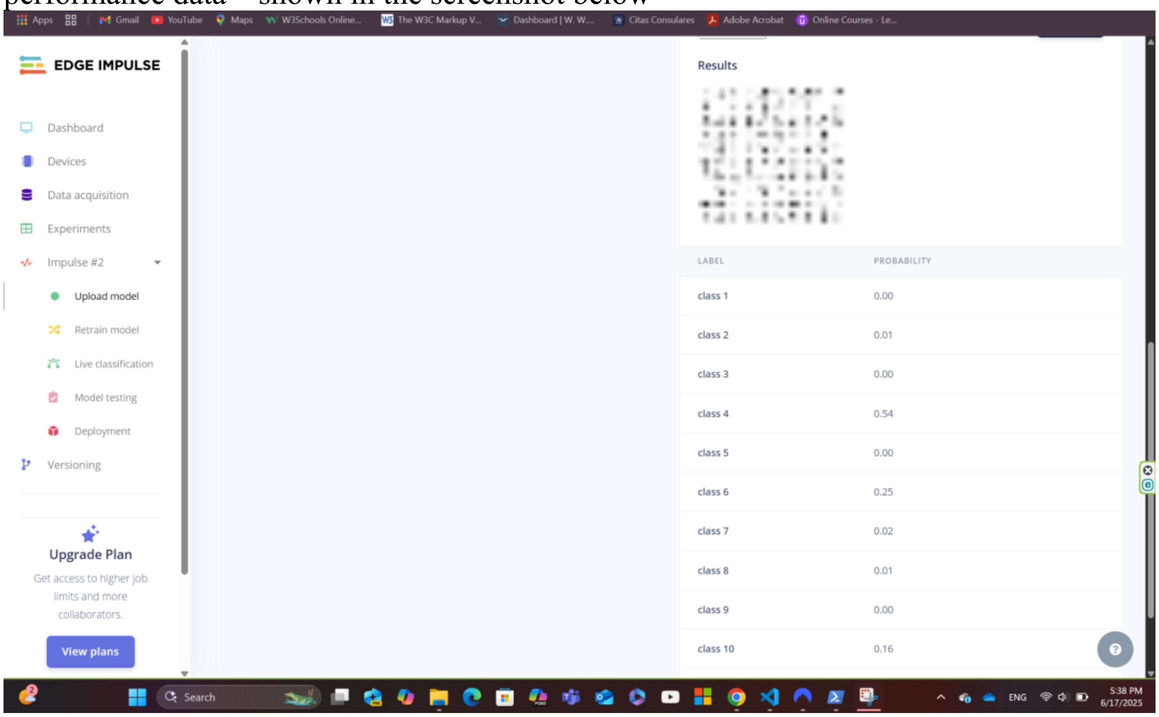
Inference Conventions

Following a successful Edge Impulse upload of the transformed TensorFlow Lite model, I tested it with the Live Classification tool included on the platform. By means of supplied sample data, this tool enables real-time testing of a model, therefore imitating the performance of the model in an actual deployment context.

I supplied a 28x28 grayscale picture of a handwritten digit from the MNIST collection for this test. The algorithm produced a prediction of class 4 with a confidence score of 54% from the image, which portrayed number 4. The model maintained its generalizing capacity after being transformed to the TensorFlow Lite version, therefore even if the confidence score was modest, the prediction was accurate.

Furthermore, the platform revealed that the model performed effectively on simulated edge devices by offering performance measures including inference time and resource use. This verified both the model's practicality for real-time embedded hardware decision-making and its accuracy.

The classification output—including the predicted label, confidence score, and performance data—shown in the screenshot below



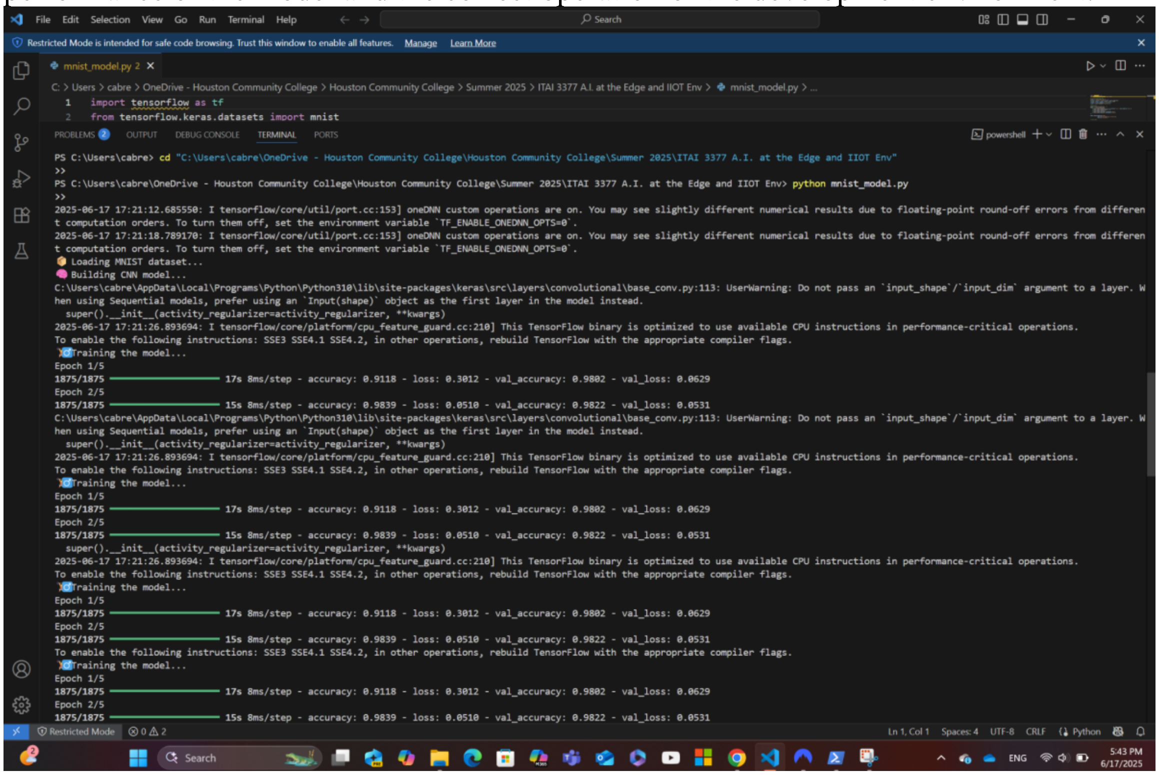
Virtual Code Model Training

The model training procedure carried out inside Visual Studio Code is seen below in a screenshot. Using a CNN developed using TensorFlow and Keras, the training was conducted on the MNIST dataset. Over five epochs, the model was taught to categorize 28x28 grayscale images of handwritten numerals ranging from 0 to 9.

For both the training and validation datasets, the terminal showed live updates on important performance measures—including loss and accuracy—all through the training process. The model showed good generalization to unseen data by the last epoch by reaching a validation accuracy of roughly 98.2%. The loss values dropped gradually over epochs, implying that the model was learning efficiently free from any evidence of overfitting.

This phase was crucial to verify proper configuration of the data preparation and architectural elements. Strong performance of the model prepared the path for effective conversion to TensorFlow Lite and later Edge Impulse platform deployment.

The screenshot offers a graphic record of the training logs, thereby verifying the good performance of the model and the correct operation of the development environment.



Environment Setup: TensorFlow Installation

The TensorFlow package and its required dependencies were effectively installed in the local Python environment with pip package manager, as the screenshot below attests to. Creating the development environment for model building and training depends critically on this phase.

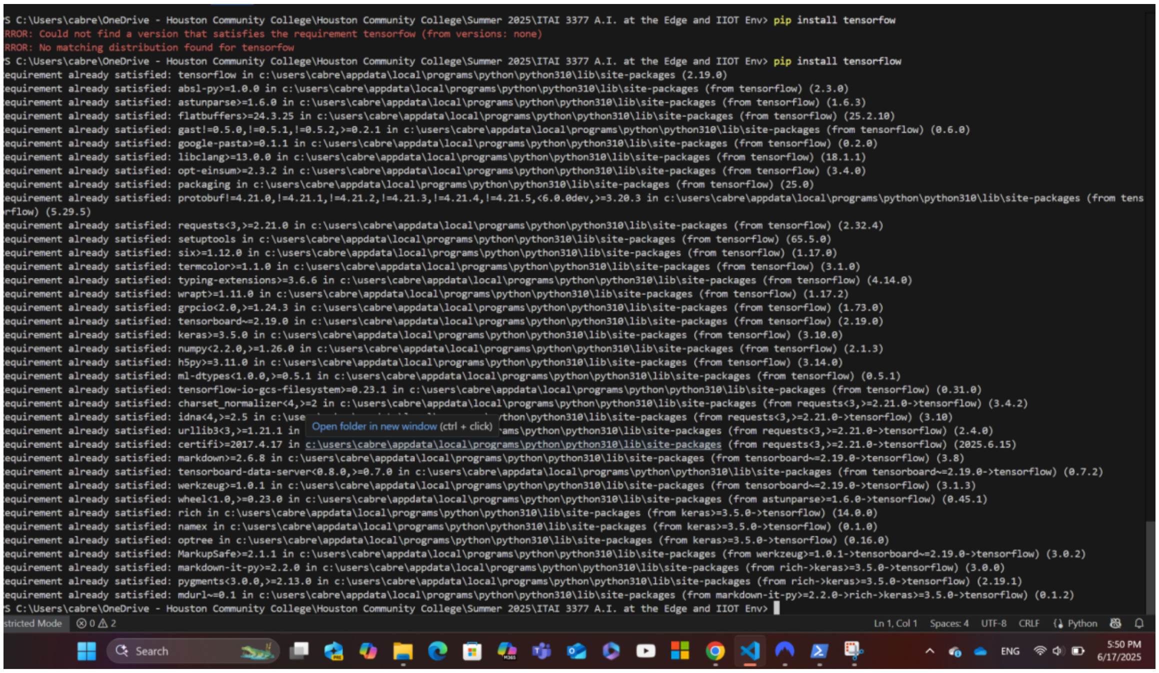
The installation method consisted in running the command:

```
pip install tensorflow
```

Once running, the terminal showed a sequence of messages indicating the installation and configuration of several TensorFlow components—including NumPy, protobuf, keras, and other necessary packages. Accessing high-level APIs used for loading data, building the neural network, training the model, and converting it to TensorFlow Lite format required correct TensorFlow installation.

This setup stage also guaranteed consistency among other dependencies, TensorFlow version, and Python interpreter. I could move boldly with coding and training the AI model in Visual Studio Code knowing the successful installation was confirmed by terminal output.

The screenshot provides proof that before development the fundamental tools for the AI pipeline were correctly set and installed.



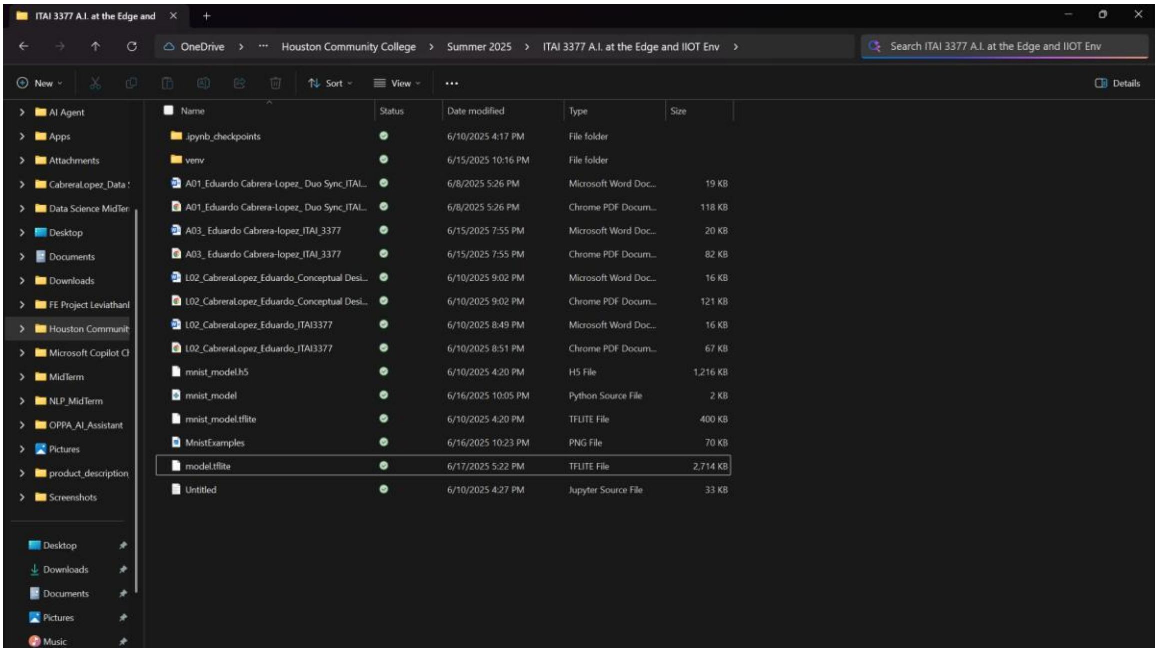
Model: Tflite File Confirmation

The successful production and presence of the model.tflite file in the working directory is confirmed by the snapshot below. Convert from Keras format to TensorFlow Lite (TFLite) using the TFLite converter, this file shows the last, refined version of the trained Convolutional Neural Network model.
Since it greatly lowers model size and increases runtime efficiency, the conversion procedure is a vital stage in getting AI models ready for use on edge devices. Specifically made to function on devices with minimal processing capability, such microcontrollers, IoT devices, and smartphones, TensorFlow Lite models are
The Python script modified the model using these lines of code:

```
converter = tf.lite.TFLiteConverter.from_keras_model(model)

tflite_model = converter.convert()
with open('model.tflite', 'wb') as f:
    f.write(tflite_model)
```

As seen in the screenshot, the model.tflite file showed up in the project directory following running this block. This verified that Edge Impulse or any other platform supporting TFLite models was ready for the model to be uploaded and used.
This phase proved not only the technical flow but also my grasp of how to move from model creation to deployment-ready forms.


ITAI 3377 A.I. at the Edge and
OneDrive > ··· Houston Community College > Summer 2025 > ITAI 3377 A.I. at the Edge and IIOT Env >
Search ITAI 3377 A.I. at the Edge and IIOT Env
New
Sort
View
Details
AI Agent
Apps
Attachments
Desktop
Documents
Downloads
Houston Communit
MidTerm
NLP_MidTerm
OPPA_AI_Assistant
Pictures
Screenshots
Music
Name
Status
Date modified
Type
Size
.ipynb_checkpoints 6/10/2025 4:17 PM File folder
venv 6/15/2025 10:16 PM File folder
A01_Eduardo Cabrera-Lopez_ Duo Sync_ITAI... 6/8/2025 5:26 PM Microsoft Word Doc... 19 KB
A01_Eduardo Cabrera-Lopez_ Duo Sync_ITAI... 6/8/2025 5:26 PM Chrome PDF Docum... 118 KB
A03_ Eduardo Cabrera-lopez_ITAI_3377 6/15/2025 7:55 PM Microsoft Word Doc... 20 KB
A03_ Eduardo Cabrera-lopez_ITAI_3377 6/15/2025 7:55 PM Chrome PDF Docum... 82 KB
L02_CabreraLopez_Eduardo_Conceptual Desi... 6/10/2025 9:02 PM Microsoft Word Doc... 16 KB
L02_CabreraLopez_Eduardo_Conceptual Desi... 6/10/2025 9:02 PM Chrome PDF Docum... 121 KB
L02_CabreraLopez_Eduardo_ITAI3377 6/10/2025 8:49 PM Microsoft Word Doc... 16 KB
L02_CabreraLopez_Eduardo_ITAI3377 6/10/2025 8:51 PM Chrome PDF Docum... 67 KB
mnist_model.h5 6/10/2025 4:20 PM H5 File 1,216 KB
mnist_model 6/16/2025 10:05 PM Python Source File 2 KB
mnist_model.tflite 6/10/2025 4:20 PM TFLITE File 400 KB
MnistExamples 6/16/2025 10:23 PM PNG File 70 KB
model.tflite 6/17/2025 5:22 PM TFLITE File 2,714 KB
Untitled 6/10/2025 4:27 PM Jupyter Source File 33 KB