Image Classification for American Sign Language

Joshua Meppiel
University of Missouri St. Louis

Abstract
This is the final project report in the Spring 2023 Deep Learning course at University of Missouri St. Louis. For this project, I have taken the image classification model I have built in the semester project and utilize it for inference on an edge device. My image classification model is capable of inferring the characters in the American Sign Language (ASL) into the English language alphabet. It takes as input an image of the human hand displaying an ASL character and predicts which character from the English alphabet is being signed. I have converted and compiled the model to run on the OAK-D Lite, also known as the OpenCV development kit.

Contents

1 Introduction

In this project, I will take my machine learning model that is capable of classifying images of the American Sign Language (ASL) alphabet and run it on the OAK-D Lite camera module[3]. This module is specifically built to run deep learning models in hardware in an optimized and performant way. This will involve taking my model that was built in TensorFlow 2 and Keras, then trained in my local lab on my own hardware, and converting it to run on the camera module.

This project is a culmination of taking the CMP SCI Deep Learning course in Spring 2023, attending Dr. Adhikari's Deep Learning Workshop at the end of 2022, and reading through the textbook Deep Learning with Python[5]

1.1 Report Layout

The remainder of this report will be structured as follows:

- Section 2: Methods
- Section 3: Converting and Compiling the Model for Edge Compute
- Section 4: Testing the model
- Section 5: Issues
- Conclusion

2 Methods

2.1 Building and Training the Model

In previous reports, I detailed the building and training of the model used within this exercise. In short, a deep learning model was built in TensorFlow 2 and Keras to perform image classification. It is capable of taking images of the human hand displaying characters of the American Sign Language (ASL) alphabet and predicting which alphabetical character is being displayed.

2.2 Training Setup

To build and train the model, I have used my local lab environment. This consists of the following:

- Software:
  - Ubuntu OS running insied Windows Subsystem for Linux (WSL2)
  - TensorFlow 2
  - Keras
- Hardware:
  - AMD Ryzen 7 5900X CPU
  - Nvidia RTX 3060Ti GPU

2.3 Inference Setup

To perform inferences, I have used my local lab environment. This consists of the following:

- Software:
  - Ubuntu OS running directly on laptop
  - DepthAI[1] API to interface with OAK-D camera module
  - OpenCV
- Hardware:

– OAK-D Lite Camera Module[3]. This camera module contains a visual processing unit (VPU), the Intel Movidius Myriad X. This VPU is capable of directly running deep learning models in a power efficent and perofrmant way. It is intended to be used in edge devices such as this OAK-D Lite camera.

2.4 Code

Note that all code and example images for this project are found in my GitHub repo[2]. Specifically, for this final edge inference report, the files 'Final Project.py', 'test.jpg', and '/model/asl.blob' are relevant.

They are described as follows:

- 'Final Project.py' : The script performing inferences in real time on the OAK-D Lite camera module.
- 'test.jpg' : The example image showing an inference of the letter 'R' in the ASL.
- 'asl.blob' : The model itself, converted to run on the Movidius Myriad X VPU.

3 Converting and Compiling the Model for Edge Compute

To convert the model for running on the OAK-D Lite camera module, it requires conversion and compilation steps. The Video Processing Unit (VPU) inside the OAK-D Lite is the Intel Myriad X processor. This processor requires models to be converted and compiled into its native format for ineference.

The OpenVINO toolset[6] is provided by Intel to perform the steps of first converting the model and then compiling it.

The steps are generally as follows:

- Save the model via TensorFlow after training is completed.
- Convert the model from TensorFlow format into ONNX format. This produces .xml and .bin files representing the model structure and its weights, respectively.
- Compile the ONNX formatted model into a native blob to run on the Movidius X processor.
- Run the model in its blob format for inference on the OAK-D Lite camera.

4 Testing the model

To test the converted and compiled model, I utilized a sample script from the Depth AI project. This script runs a model on the OAK-D camera module. When inferences are performed successfully, a bounding box is drawn around the character in question. I ran this script and displayed various characters of the ASL in front of the camera. It was able to pickup some of them and draw bounding boxes around them. In **Figure** 1 I show an example of displaying the letter 'R' in the ASL alphabet. The neural network is able to identify this character and report the bounding box XY coordinates. OpenCV is then used to draw the bounding box around the character.

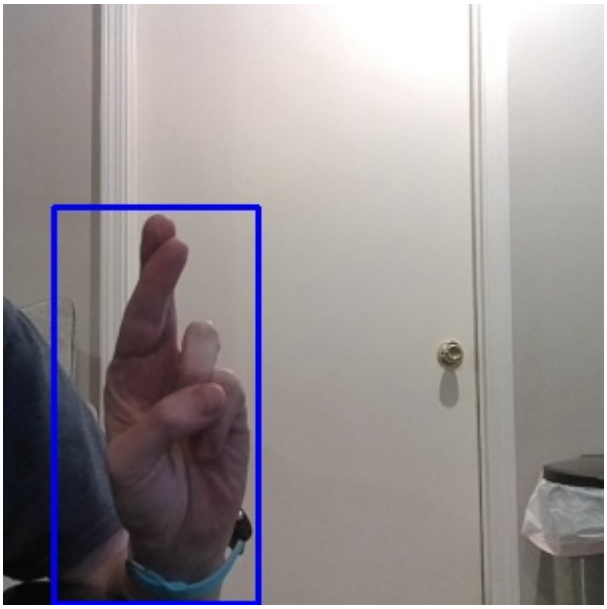

Figure 1: The letter 'R' in ASL.

5 Issues

I ran into multiple issues converting the model to run on the OAK-D camera.

First, the default method for TensorFlow 2 to save a model is into a .pb formatted file with separate variables. The model optimizer tool provided by OpenVINO produced errors when attempting to convert these models. I instead moved to using the Open Neural Network Exchange[4] format. This is an open standardized format for neural networks that can be used to run inferences on a variety of hardware. I used the ONNX Python package to save my TensorFlow 2 model in ONNX format. This produces both .xml and .bin files that represent the model structure (.xml) and model weights (.bin). I then utilized the OpenVINO compilation tool to build the model into a blob file that can be loaded onto the camera, which brings me to my next problem.

The second issue I encountered was that the model compilation tool would build the model to expect an incorrectly formatted input image. After some searching through the bugs reported to the project's GitHub site, I was able to arrive at a satisfactory set of options in the compilation tool that would allow my model to run on the edge device.

6 Conclusions and Future Work

I found my engineering exercise of converting, compiling, and then running my model on an edge compute device designed for AI inference to be a useful and fun learning experience. It gave me real world practical experience in how a model may be deployed and utilized.

For future work on this topic, I'd like to enhance the usage of the ASL model to display the characters being signed; at the moment it only draws bounding boxes around them.

References

[1] Depthai.

[2] Github repo.

[3] Oak-d lite.

[4] Open neural network exchange.

[5] Francois Chollet. *Deep Learning with Python*. Manning Publications Co., USA, 1st edition, 2017.

[6] Intel. Openvino.