International Journal of Scientific & Engineering Research, Volume 14, Issue 10, October–2023
ISSN 2229–5518

Setting Up Apple M Series Macbooks to Work with OpenCV and Java

Susmit Sarkar

Abstract— Apple Macbooks are now enabled with Apple M1 or M2 processors, and because of the architectural differences, existing libraries like OpenCV are not quite easy to use with Java on Apple M1 or M2-powered Macbooks. This paper takes an investigative approach to figure out those potential issues and comes back with a potential solution. This follows an approach of making OpenCV Jar locally before importing and using it in IDEs.

Index Terms— Setting up Apple M1 Macbook, OpenCV with Java, M1 Macbook with OpenCV and Java, OpenCV Java Set up on M2 Macbooks

1 Introduction

OpenCV is an open-source computer vision library published by Intel under the Apache 2 licence, mainly for the purpose of real-time computer vision. It was developed in 2006 using C++, but most of the major development work was carried out in 2009. And since then, mostly minor releases have been done, and OpenCV hasn't considered including much of the machine learning features. In 2020, OpenCV announced a kickstarter campaign for the OpenCV AI Kit, a series of hardware modules and additions to OpenCV supporting spatial AI.

Though the library was initially developed using C, most of its newer developments and algorithms are now developed using C++. But it provides language binding in multiple other languages, including Java.

It is designed in such a way that if the library finds Intel's Integrated Performance Premitives on the system, it will use these proprietory-optimised routines to accelerate itself. A Compute Unified Device Architecture (CUDA)-based graphics processing unit (GPU) interface has been in progress since 2010, and there have been a few major milestones achieved.

Macbooks used to use Intel's x86 processors, and hence OpenCV used to be integrated well with those processors. Apple started using ARM-based M1 and M2 chips for its newer Macbooks. Though it runs x86-based applications as well using Rosetta (which could cause adverse effects on performance when running x86 applications), fortunately, due to its speed, the M1 still outperforms older Intel chips in most scenarios, even with legacy x86 apps.

OpenCV is one of those products impacted by this but fortunately this paper describes a way to bypass this until a more viable solution is being put into place.

2 Issues Running OpenCV on M Macbooks

2.1 How ARM is different to x86

Apple Macbooks started using it's proprietory M1 and M2 chips instead of using Intel produced microprocessors. This uses ARM architecture. ARM is a family of RISC instruction set architecture for computers. Intel used to produce 32-bit microprocessors (successors of 16-bit 8086 microprocessor) and mostly they were used by Apple, Microsoft and other providers on laptops and computers.

ARM uses Reduced Instruction Set Computing (RISC), while x86 uses Complex Instruction Set Computing (CISC).

RISC has a far fewer instructions than CISC, and each basic instruction is executed in a single clock cycle. CISC's instructions can be pretty complex and perform multiple tasks in a single instruction. CISC's complex instruction set makes x86 chips harder to design because the chip has to be able to account for the complex instructions, making x86 chips typically more expensive.

But x86 applications can't directly run on ARM as x86 uses CISC instruction set. Developers at Apple designed Rosetta 2 that allows older x86 applications to run on newer M Series chips.

x86 processors typically operate independently of peripheral components like RAM and GPUs. But ARM microprocessors were designed to package these additional components into a combined central unit. That's why ARM processors operate as part of a System on Chip (SOC).

2.2 Why OpenCV won't work directly

OpenCV focuses on real time image processing, video capture and analysis. To serve this OpenCV libraries need to interact with CPUs and GPUs through complex instruction sets. x86 used to accept those instructions and process easily. But this will be a bit difficult to be processed by ARM as it will need reduced set of intructions.

So, unless these compiler instructions can be emulated for RISC system, OpenCV can't directly work on Apple M Series Macbooks.

OpenCV is yet to publish an official one click downloadable build to support this.

3 System Specifications

M1 Series Apple Macbook Air was used for this alongside some other packages those were brewed to specific versions *(Details provided in Section 4)*. IntelliJ IDEA 2023.1.1 was used for demonsration purpose. This demonstration will use OpenCV 4.5.0 but the similar set up should work for other versions as well. With IntelliJIDEA Amazon Corretto 21.0.1 distribution of JDK was used for demonstration purpose but it can be any distribution that supports arm64.

4 Setting Up M Series Macbook for OpenCV

4.1 Setting up FFMPEG4

FFmpeg is an open-source software that internally gets used by OpenCv to handle video, audio and other multi-media formats. OpenCV is compatible with FFmpeg 4 and below at the moment. It is vital to check if the Macbook is having any higher versions installed and those needs to be uninstalled before installing FFmpeg 4.

```
brew install FFMPEG@4
brew link FFMPEG@4
```

4.2 Install Support Tools

The approach here will be to use a compiler independent method to build, test and package the OpenCV for M series macbooks ideally in form of a JAR file. CMake will be used here to exactly do that. It's not a build system itself but it generates another system's build files. It is usually used in conjuction with native build libraries like Make or XCode.

Apache Ant on the other hand is a Java based build tool similar to Make that will also be used to build the library for Java.

Also, AdoptOpenJDK will be needed to be installed for Java support. Also, it's crucial that $JAVA_HOME is set at this point as that will be needed during CMake is used.

```
xcode-select --install
brew install adoptopenjdk
brew install cmake
brew install ant
```

4.3 Download OpenCV

Any source control system can be used to download the project but here wget has been used. The project is downloaded and unzipped in a directory. Alongside this method, the project can also be downloaded using git or other version control tools. Here 4.5.0 has been used as a reference but any future versions should follow same process.

```
brew install wget
wget -O opencv.zip
https://github.com/opencv/opencv/archive/4.5.0.zip
unzip opencv.zip
cd opencv-4.5.0
mkdir build && cd build
```

4.4 Building the project with Java option

In this step, finally CMake can be used to make the build from downloaded repository. System Processor and OSX Architecture levels need to be set up as "arm64" for M series Macbooks. It is also crucial to use "amd64" version of Java AWT Library and "arm" version of JVM Library.

Finally, BUILD_opencv_java flag should be "ON" in terms of making sure that it generated JAR file and Java library.

```
cmake -DCMAKE_SYSTEM_PROCESSOR=arm64 \
-DCMAKE_OSX_ARCHITECTURES=arm64 \
-DWITH_OPENJPEG=OFF \
-DWITH_IPP=OFF \
-D CMAKE_BUILD_TYPE=RELEASE \
-D CMAKE_INSTALL_PREFIX=/usr/local/opencv \
-D JAVA_INCLUDE_PATH=$JAVA_HOME/include \
-D
JAVA_AWT_LIBRARY=$JAVA_HOME/jre/lib/amd64/libawt.so \
-D
JAVA_JVM_LIBRARY=$JAVA_HOME/jre/lib/arm/server/libjvm.so \
-D BUILD_opencv_python2=OFF \
-D BUILD_opencv_java=ON \
-D INSTALL_PYTHON_EXAMPLES=OFF \
-D INSTALL_C_EXAMPLES=OFF \
-D OPENCV_ENABLE_NONFREE=OFF \
-D BUILD_EXAMPLES=ON ..
```

In an ideal scenario, this make should not shout any errors. In case of errors, it's ideal to make sure that the right version of JDK is installed and $JAVA_HOME is set up correctly.

Make is a build automation tool that builds executable programs and libraries from source code by reading files called makefiles which specify how to derive the target program. This will be used to make the executables at this point.

```
make -j8
```

4.5 Installing the build

If this build is successful without any errors, installation can be kick started at this point.

```
sudo arch -x86_64 make install
```

Based on the architecture type, this might need to be installed

differently.

```
sudo make install
```

5 Testing The Success

5.1 Checking files

As on Section 4.4, CMAKE_INSTALL_PREFIX was defined and ideally this is where inside "../share/java/opencv4" the Jar file and the native library will be placed. If these files are there, mostly the install was successful.

5.2 Setting up IntelliJ IDEA

A Maven project can be created to run a basic set up check. The POM file should be having the dependancy added:

```
<dependency>
  <groupId>org.openpnp</groupId>
  <artifactId>opencv</artifactId>
  <version>3.4.2-0</version>
</dependency>
```

The generated OpenCV JAR and library should be added to the project library from the "Module Settings". 2 files should be added in "Libraries" option:

1. opencv-450.jar

2. libopencv_java450.dylib

These file names could different based on the targetted OpenCV version.

5.3 Setting up IDE level Java distribution

The project should be using the right Java distribution and version to match the arm64 architecture. This was tested with Amazon Corretto Version 21.0.1. Java distribution can be changed in IntelliJ from the "Module Settings" as well from "Modules" option.

5.4 Writing basic Java code to check the set up

A basic Java class can be created to check if the targetted JAR has been built in the right way.

```
package com.susmit.opencv;
import org.opencv.core.Core;
public class MainLibrary {
   public static void main(String[] args) {
     System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
     System.out.println("Loaded OpenCV version "+
Core.VERSION);
   }
}
```

As this is having a main class, this can be executed straight away from the IntelliJ IDEA. The outcome should shout the right OpenCV version like this:

```
Loaded OpenCV version 4.5.0

Process finished with exit code 0
```

6 Conclusion

As Apple M Series Macbooks are quite new into the market, it will take some time of developers to realign their applications so that they can be seamlessly used in those Macbooks. Error messages are not very evidential at the moment because of the maturity of M series macs. But certain technical implementations like this will help developers work with OpenCV until a more viable and stable solution hits the market.

References

[1] OpenCV.org - Official website of OpenCV
[2] Pulli, Kari; Baksheev, Anatoly; Kornyakov, Kirill; Eruhimov, Victor. "Realtime Computer Vision with OpenCV".
[3] Adrian Kaehler; Gary Bradski. Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library. O'Reilly Media. ISBN 978-1-4919-3800-3.
[4] Hajdarbegović, Nermin. "Apple M1 Processor Overview and Compatibility"
[5] Reidt, Teresa. "ARM vs. x86: Differences & similarities of both architectures"