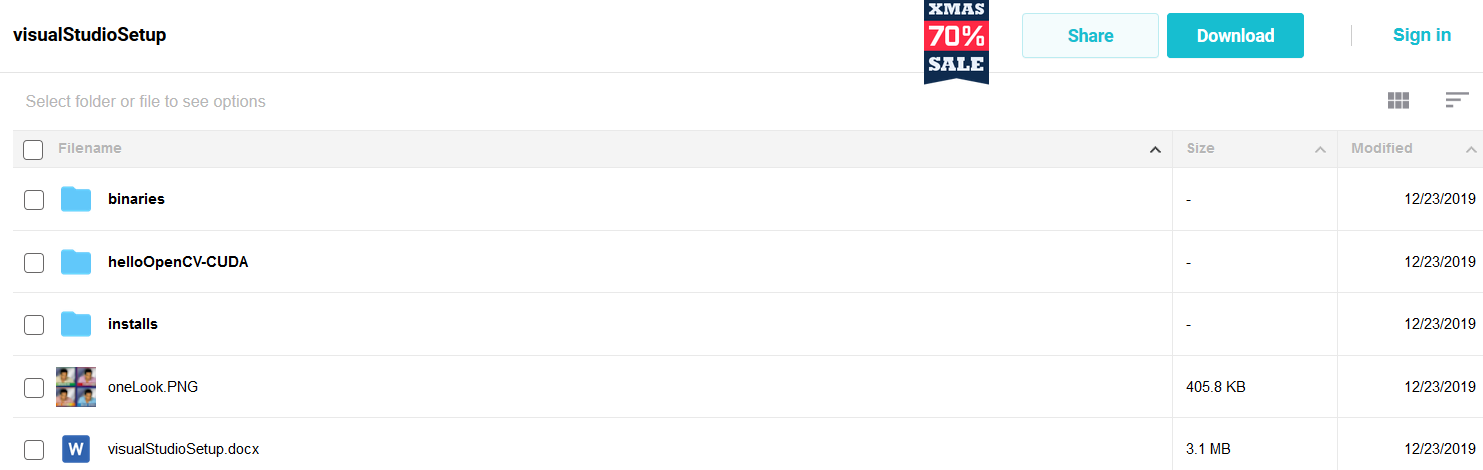
# Introduction

The goal of this document is to walk the user through the installation of Visual Studio 2019 with the following capabilities.

* C++ & Python Development with Cross-debugging
* CUDA Support
* Open CV Support with CUDA Extensions

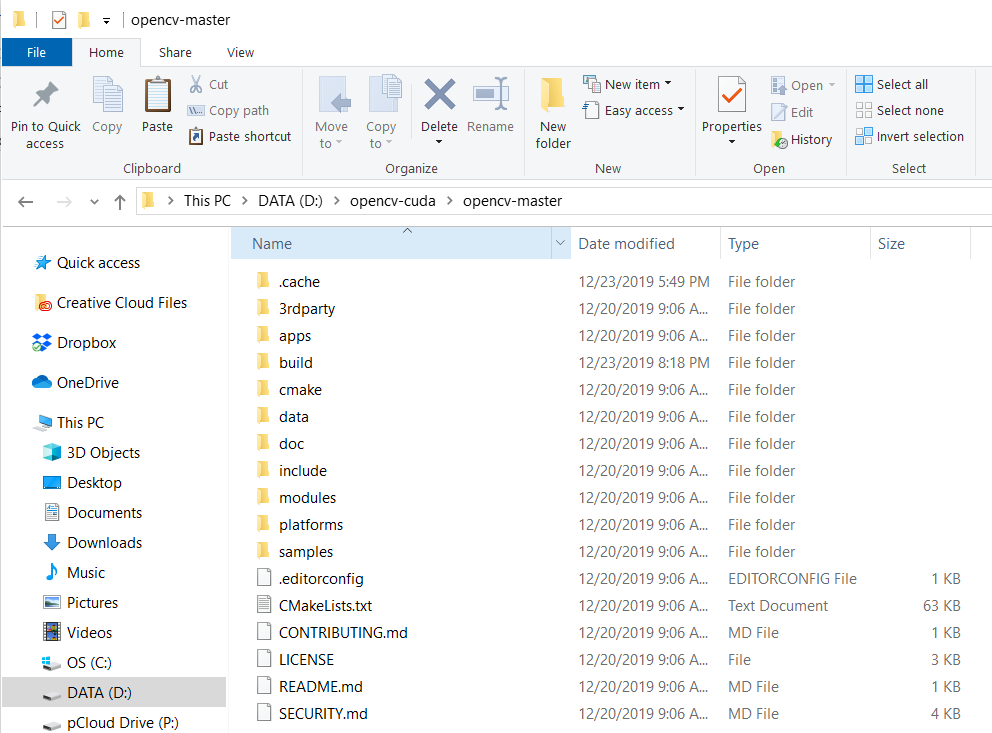
The requirements for this document are a Win 10 PC with an NVIDIA graphics card and an internet connection. Throughout this document the original links for all files will be included however all necessary files have been aggregated and are available at the below link. The password is **moog123**. Simply select all files and click on *download selected*.

|  |
| --- |
| <https://my.pcloud.com/publink/show?code=kZHrHSkZgWzngHSvrFm5vWKFspVwwutlg1n7> |





Once you have downloaded all the files, save them in a directory like D:\opencv-cuda. Pay close attention to paths as they are key to this installation. Make sure that there are not double directories which happens when one does an unzip sometimes. Verify that the path to the directories inside opencv-master is located in the same directory as shown below.

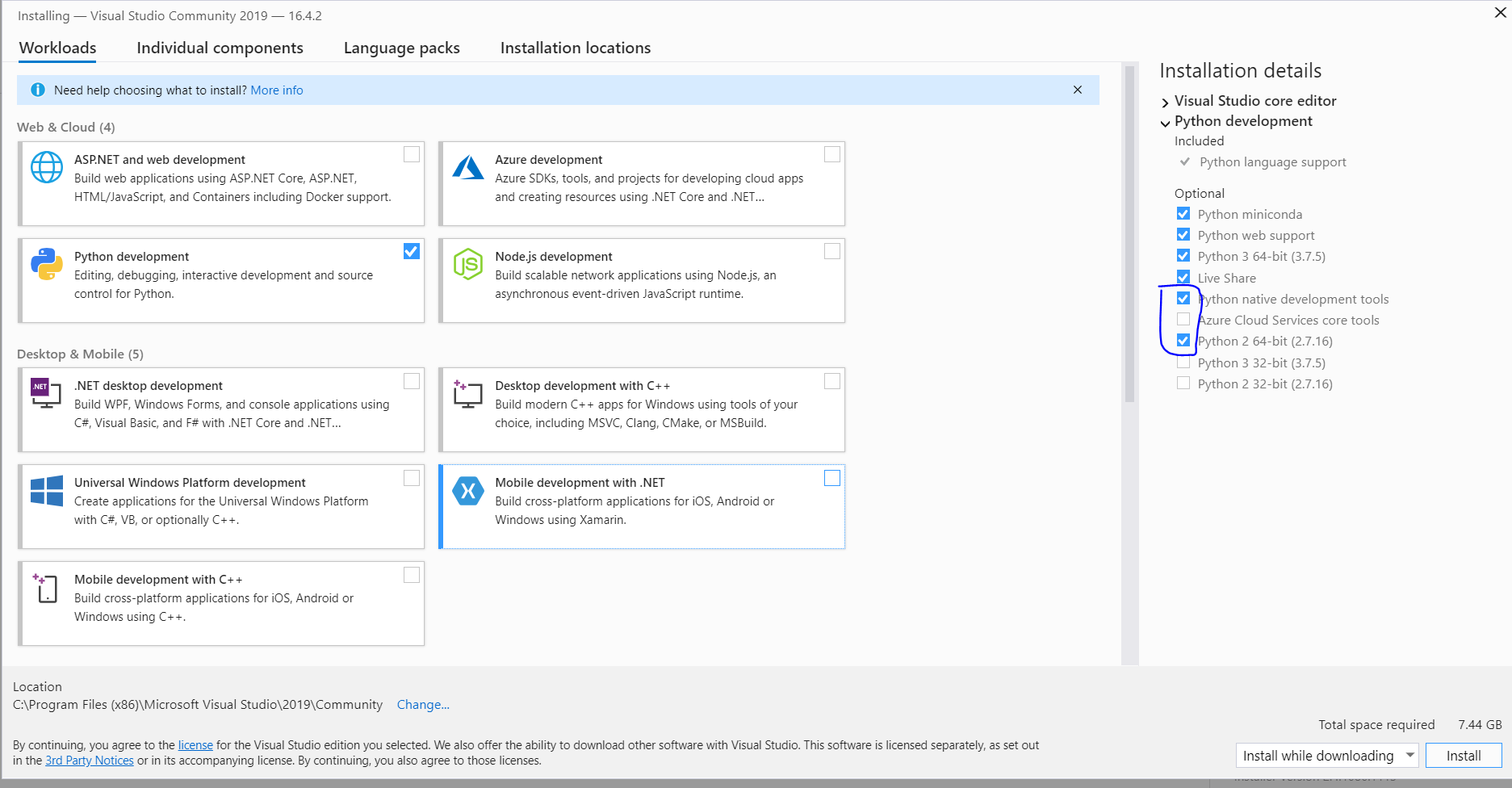


# Install Microsoft Visual Studio 2019

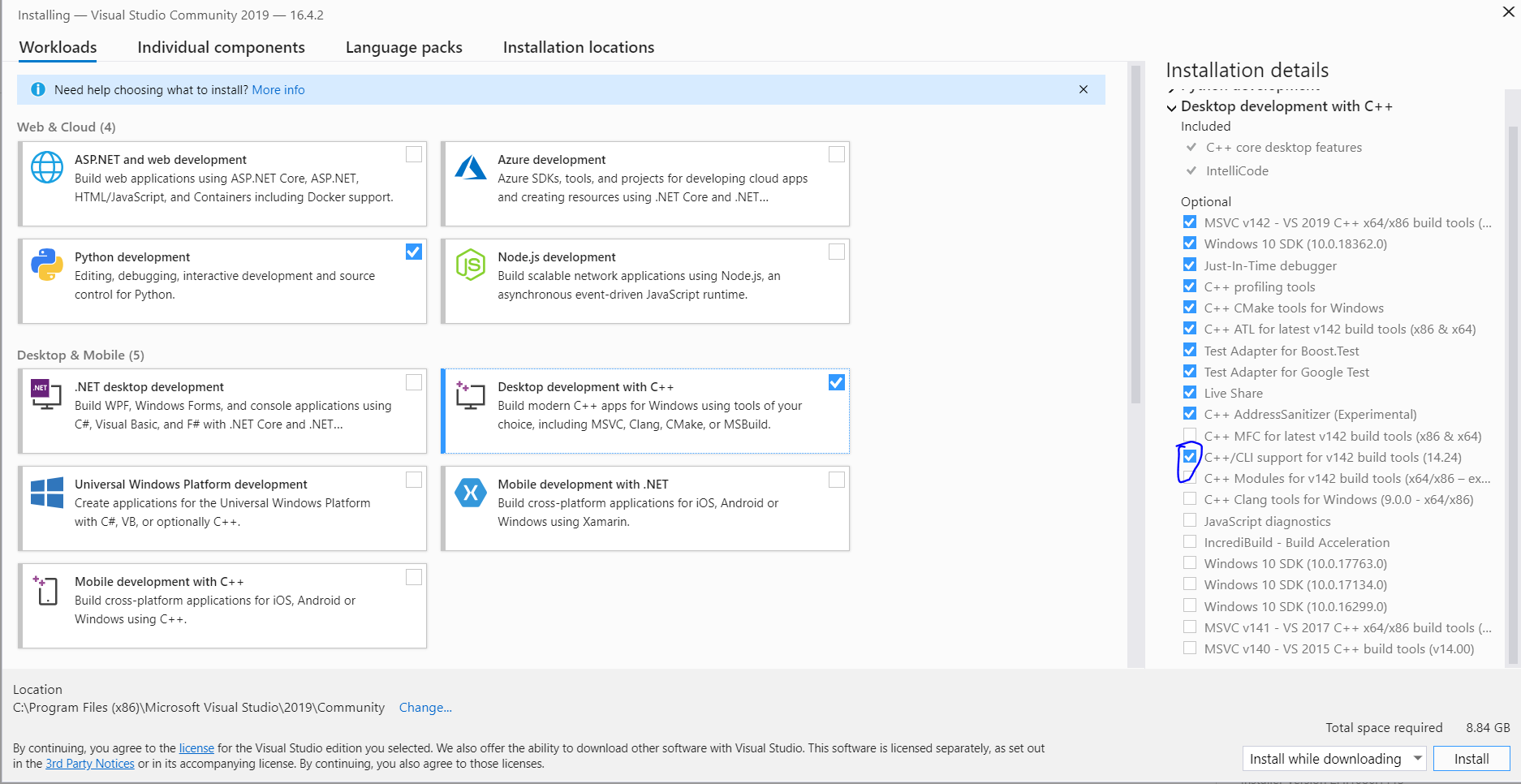
Download the Visual Studio 2019 Installer from the link below and click on it to start the installation. Alternatively just use the ‘vs\_community\_....exe’ file from the aggregated code base above.

<https://visualstudio.microsoft.com/thank-you-downloading-visual-studio/?sku=Community&rel=16>

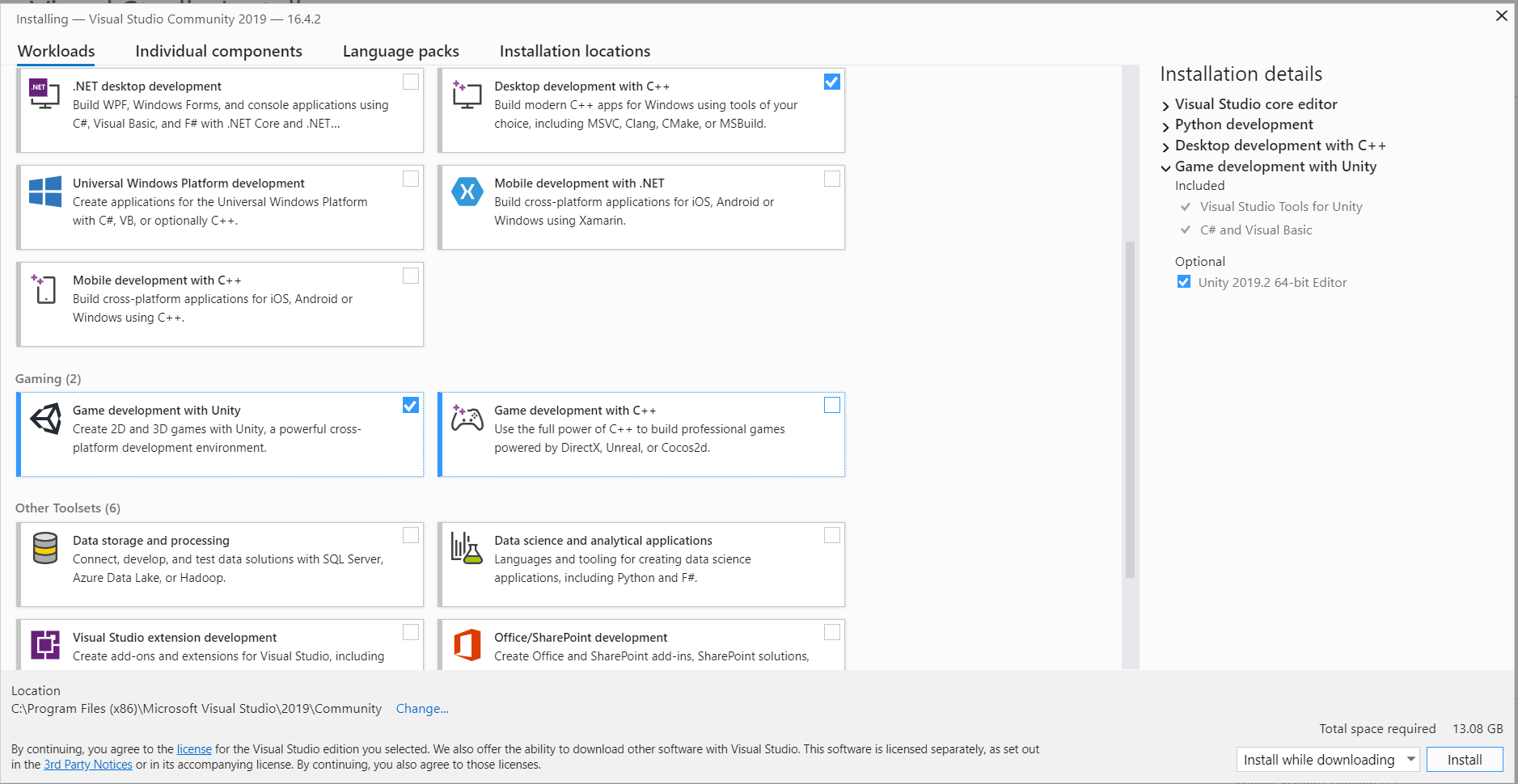
If you want python support make sure to click on the python development box and also check the two boxes that are circled as they are not part of the default installation.

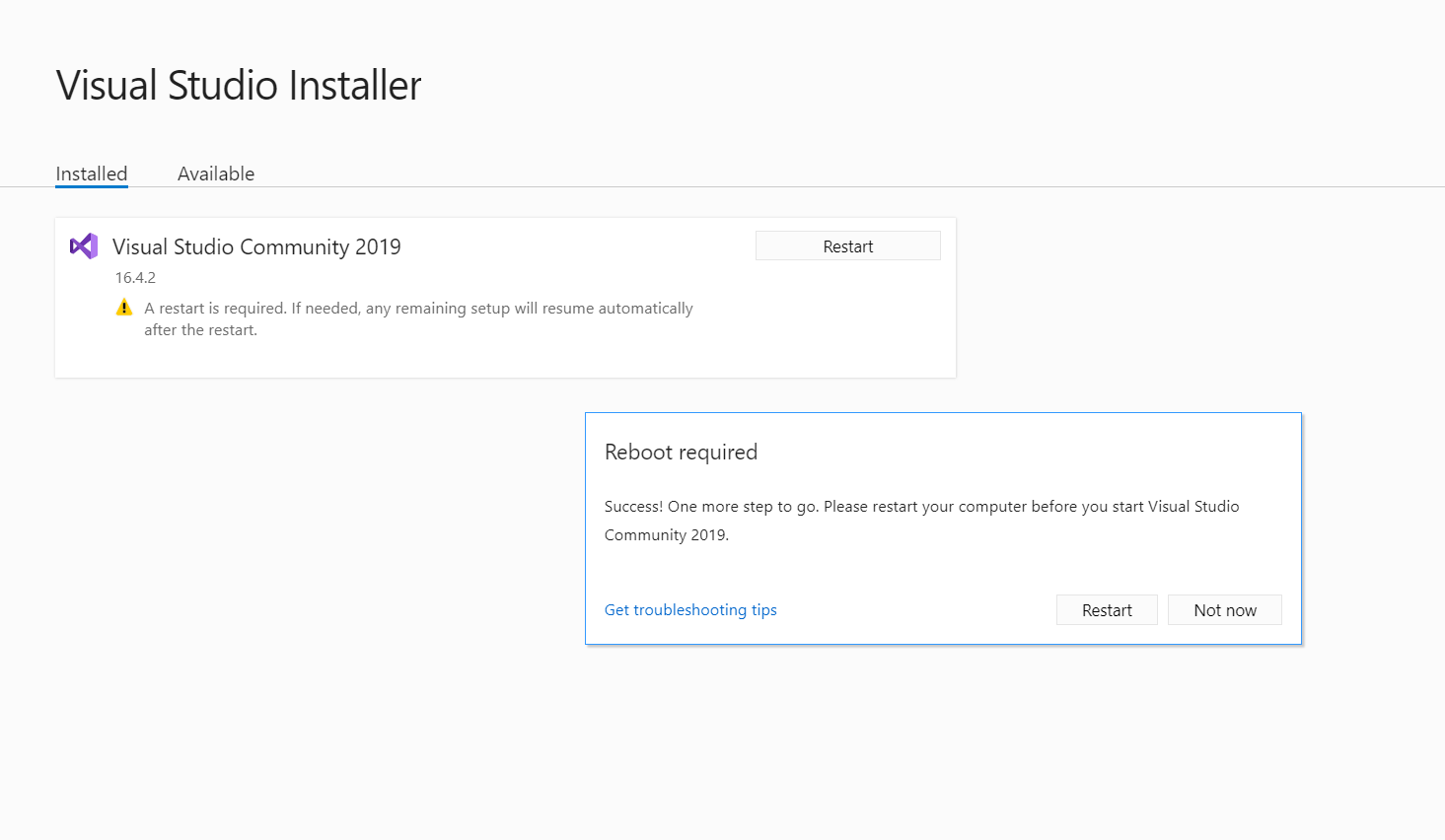


For C++ applications make sure to check the C++ development tab and click on the CLI [command line interface] check box as shown below as it is not part of the default installation.



Another installation option is to install Unity support which can be useful for developing Unity applications for creating synthetic photo realistic data sets for AI training applications. This may not be useful right now though and it is a large addition to the total download size.

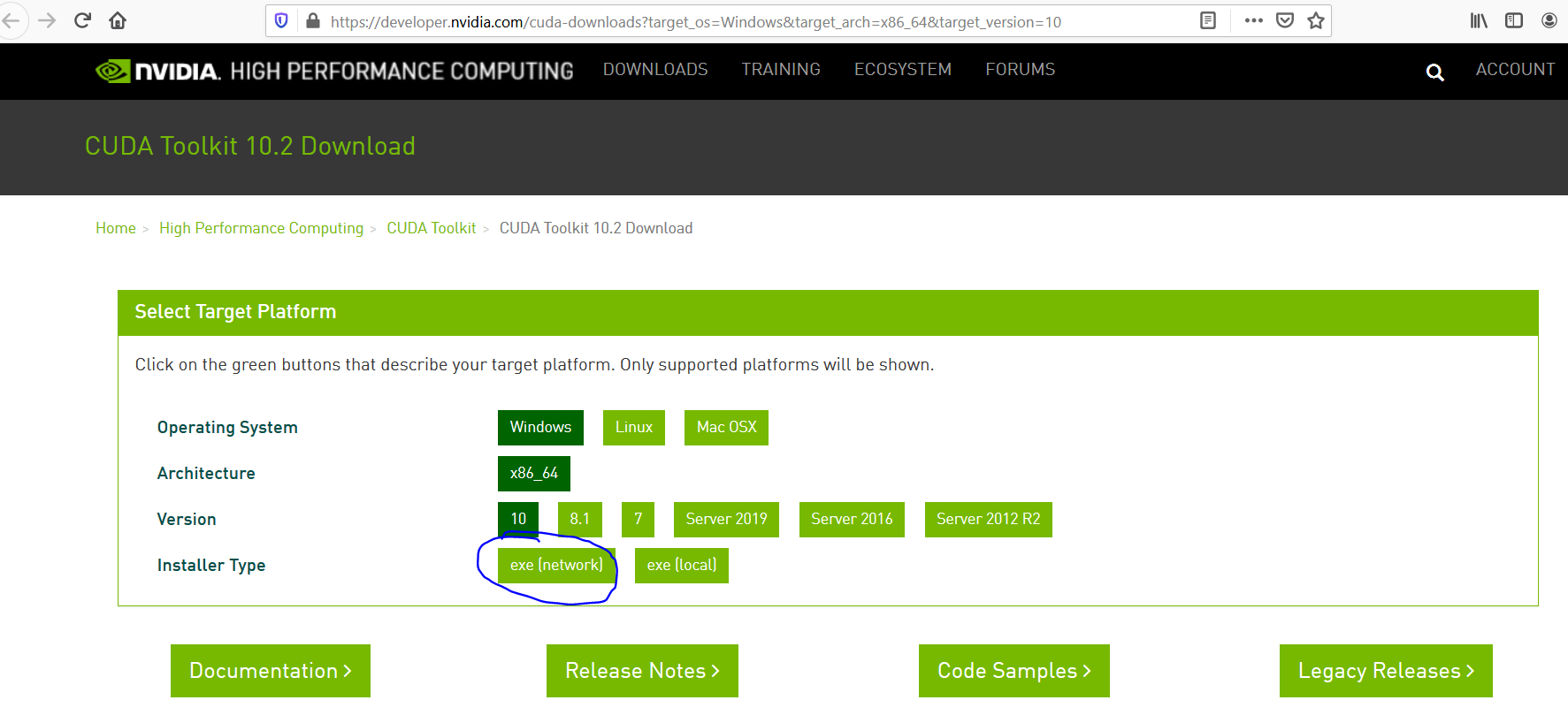




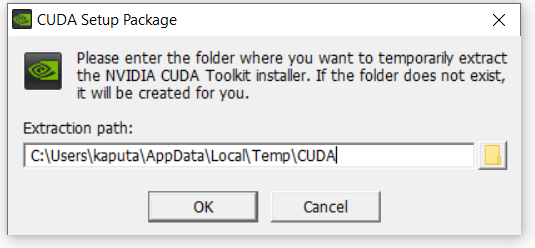
# Install NVIDIA CUDA TOOLKIT [10.2]

In order to use OpenCV with CUDA capability the correct CUDA development framework must be downloaded. The link is shown below and the proper framework is also located in the file aggregate mentioned above.

<https://developer.nvidia.com/cuda-downloads>



The temporary location for the installer doesn’t really matter so just select he default location. Click on the express installation and follow the images below.



|  |  |
| --- | --- |
|  |  |

After installing the CUDA toolkit make sure that it is located on the system path by checking out the environment variable called CUDA\_PATH as shown below.

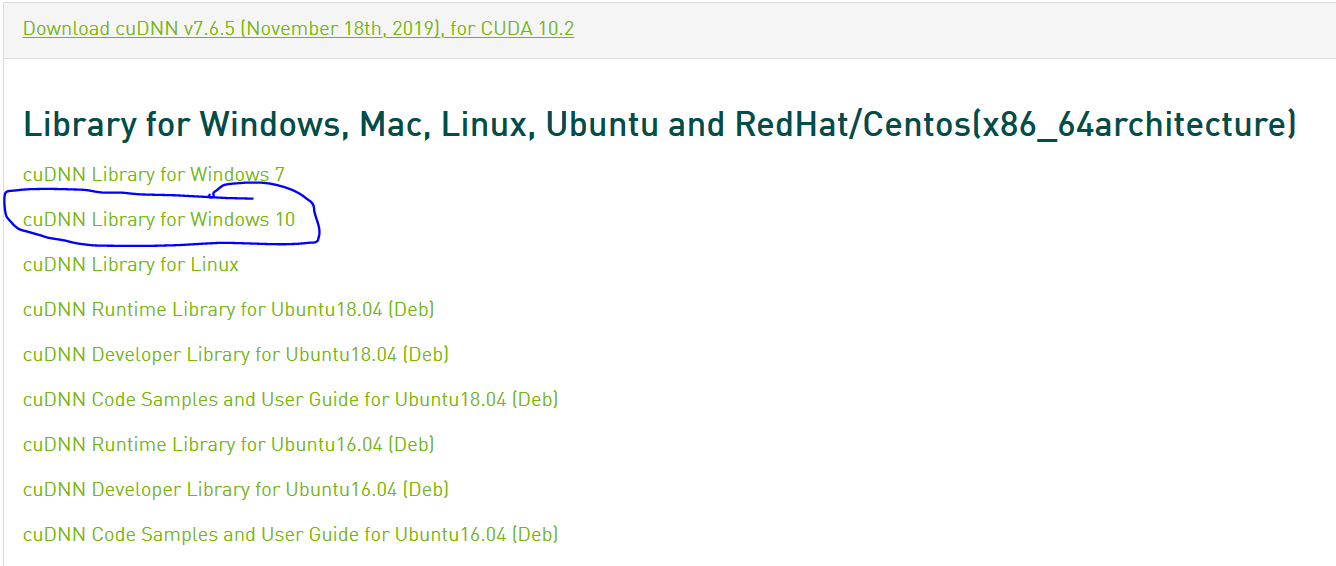
# 

# Install NVIDIA cuDNN

NVIDIA also has a wide variety of deep neural network functions optimized for their GPUs and in order to make use of these extensions one must install the cuDNN framework.

<https://developer.nvidia.com/cudnn>





After pulling down the cuDNN .zip file it must be installed by manually copying over several files. Assuming your CUDA toolkit is located at the default location you will need to copy the below files to the below locations. Note that <installpath> should be something like D:\opencv-cuda as mentioned in the beginning of this document.

Install CuDNN

|  |
| --- |
| <installpath>\cuda\bin\cudnn64\_7.6.5.32.dll to C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.2\bin |
| <installpath>\cuda\ include\cudnn.h to C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.2\include |
| <installpath>\cuda\lib\x64\cudnn.lib to C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.2\lib\x64 |

# Install NVIDIA Video Codecs

In order to decode and encode video with GPU acceleration one must install the NVIDIA video codec SDK per the link below.

<https://developer.nvidia.com/nvidia-video-codec-sdk/download>

Install video codec

|  |
| --- |
| <installpath>\Video\_Codec\_SDK\_9.1.23\ include\cuviddec.h to C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.2\include |
| <installpath>\Video\_Codec\_SDK\_9.1.23\ include\nvcuvid.h to C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.2\include |
| <installpath>\Video\_Codec\_SDK\_9.1.23\ include\nvEncodeAPI.h to C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.2\include |
| <installpath>\ Video\_Codec\_SDK\_9.1.23\lib\x64\nvcuvid.lib to C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.2\lib\x64 |
| <installpath>\ Video\_Codec\_SDK\_9.1.23\lib\x64\nvencodeapi.lib to C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.2\lib\x64 |

Similar to the cuDNN installation the video codec installation is manual and the above files need to be manually copied to the above locations.

# Download OpenCV and OpenCV-Contrib

One can simply use the directories located in the aggregate mentioned at the top of this document or the files can be downloaded via the below links.

<https://github.com/opencv/opencv>

<https://github.com/opencv/opencv_contrib>

# Install Python via Anaconda

For this installation python 2.7 was chosen for the default version however python 3 can be chosen as the default if you like. Download the installations via the link or use the files from the file aggregate. If you don’t have any other important programs that run python you can select to add anaconda to the system path as I have done below.

<https://www.anaconda.com/distribution/>

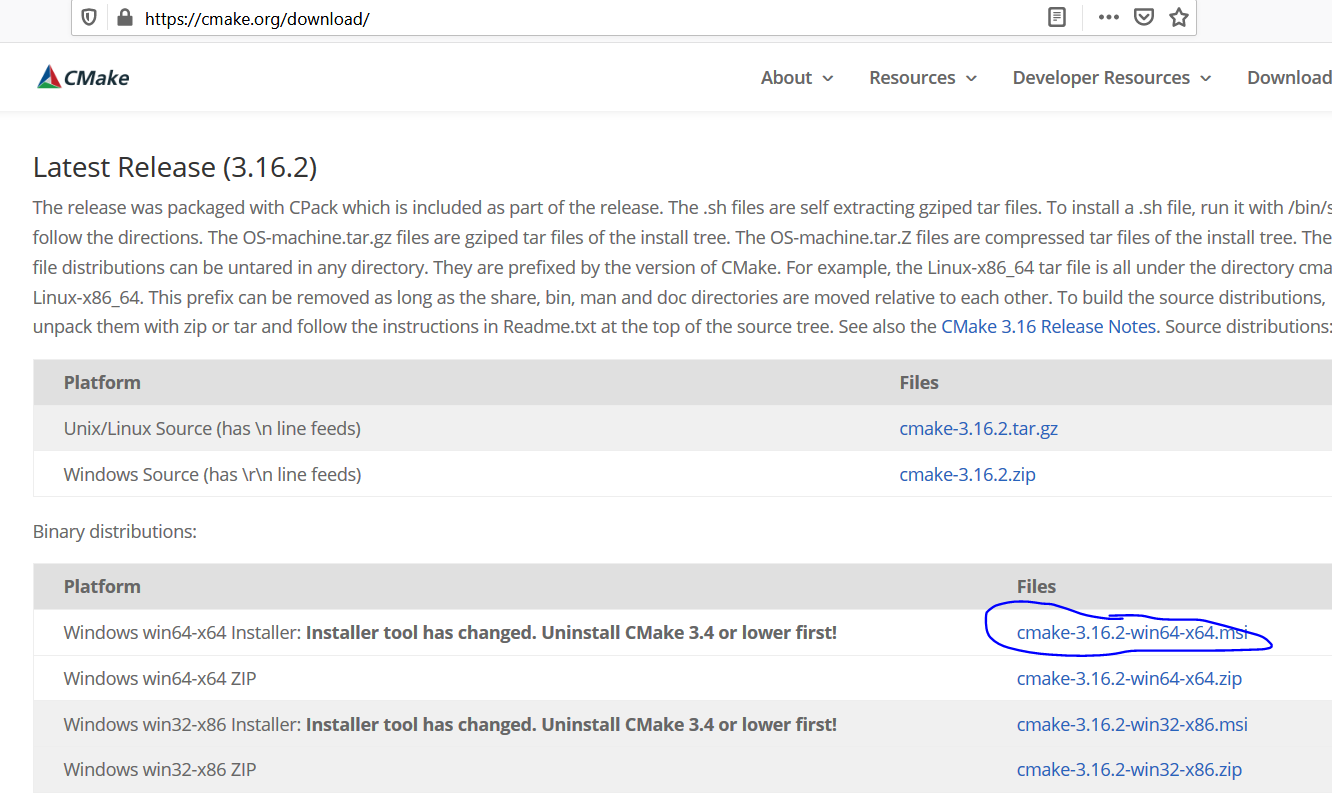
|  |  |
| --- | --- |
|  |  |

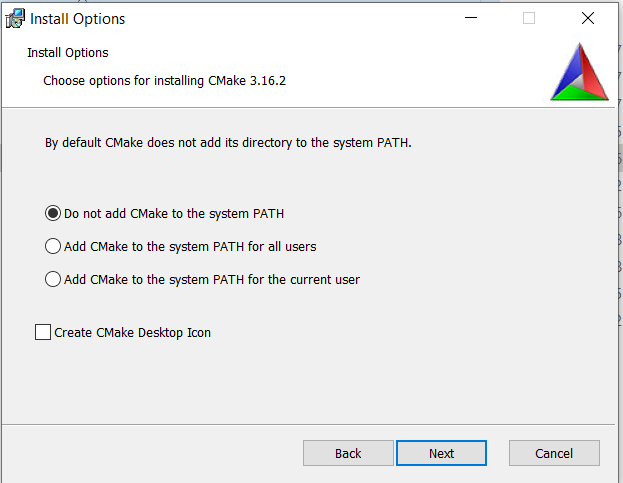
|  |  |
| --- | --- |
|  |  |

# Download CMake

Use the below link or the aggregate file.

<https://cmake.org/download/>





# Create Initial Pass at Make File

The first step to compiling OpenCV is to create a make file using cmake. Open a cmd prompt and enter the below information. Note that the paths should match where the files are located on your system. It is highly recommended to change where the file is located on the system rather than the paths below.

set "openCvSource=D:\opencv-cuda\opencv-master"

set "openCVExtraModules=D:\opencv-cuda\opencv\_contrib-master\modules"

set "openCvBuild=%openCvSource%\build"

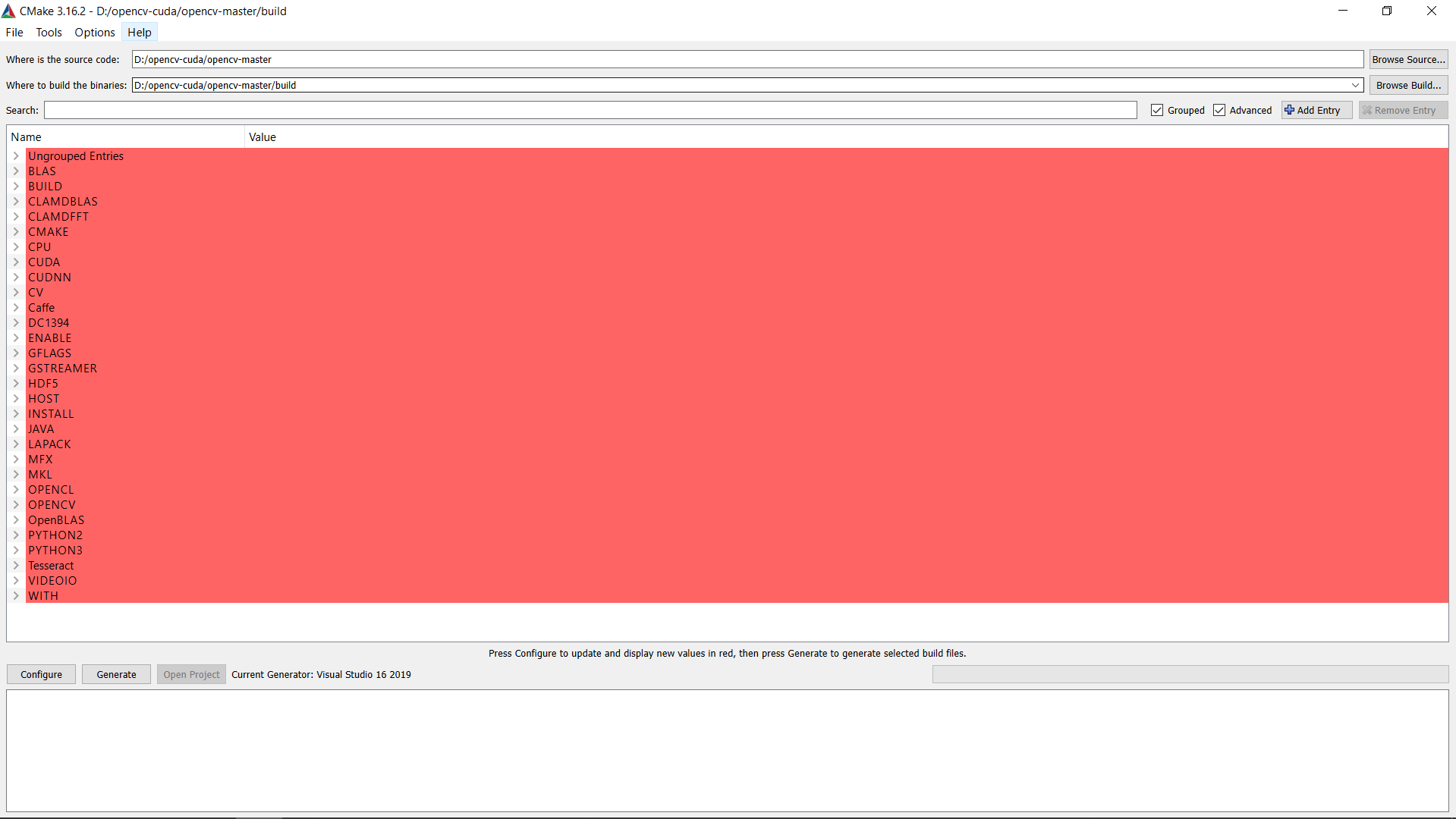
set "buildType=Release"

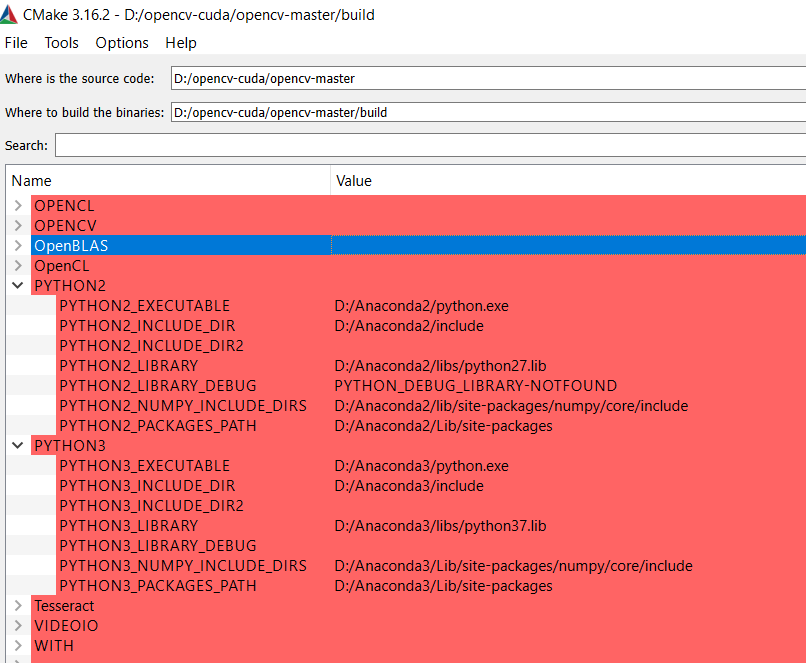
set "generator=Visual Studio 16 2019"

After entering the above lines, copy and paste in the below long line of code. This code will create a make file for the OpenCV compilation.

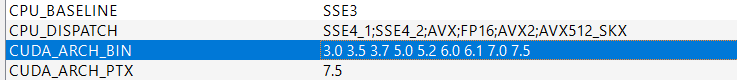
"C:\Program Files\CMake\bin\cmake.exe" -B"%openCvBuild%/" -H"%openCvSource%/" -G"%generator%" -DCMAKE\_BUILD\_TYPE=%buildType% -DBUILD\_opencv\_world=ON -DBUILD\_opencv\_gapi=OFF -DWITH\_CUDA=ON -DCUDA\_TOOLKIT\_ROOT\_DIR="C:/Program Files/NVIDIA GPU Computing Toolkit/CUDA/v10.2" -DCUDA\_FAST\_MATH=ON -DWITH\_CUBLAS=ON -DINSTALL\_TESTS=ON -DINSTALL\_C\_EXAMPLES=ON -DBUILD\_EXAMPLES=ON -DWITH\_OPENGL=OFF -DOPENCV\_EXTRA\_MODULES\_PATH="%openCVExtraModules%" -DOPENCV\_ENABLE\_NONFREE=ON -DCUDA\_ARCH\_PTX=7.5 -DWITH\_NVCUVID=ON -DWITH\_MFX=ON -DINSTALL\_PYTHON\_EXAMPLES=ON -DWITH\_GSTREAMER=OFF -DBUILD\_opencv\_python3=ON -DBUILD\_opencv\_python2=ON

After entering the above line open CMake via that windows start menu and navigate to the first pass make file. The project is located in d:\opencv-cuda\opencv-master\build and should look something like this. Make sure to select the *grouped* checkbox as shown below. **You may have to update the python paths as I did below**.





When you click on *configure* you might get an error relating to the DNN version and you will have to go to this line and delete all the numbers below 6.0. Then type *configure* again and it should return success. After that you can click on *generate* and a visual studio project should be generated for you.

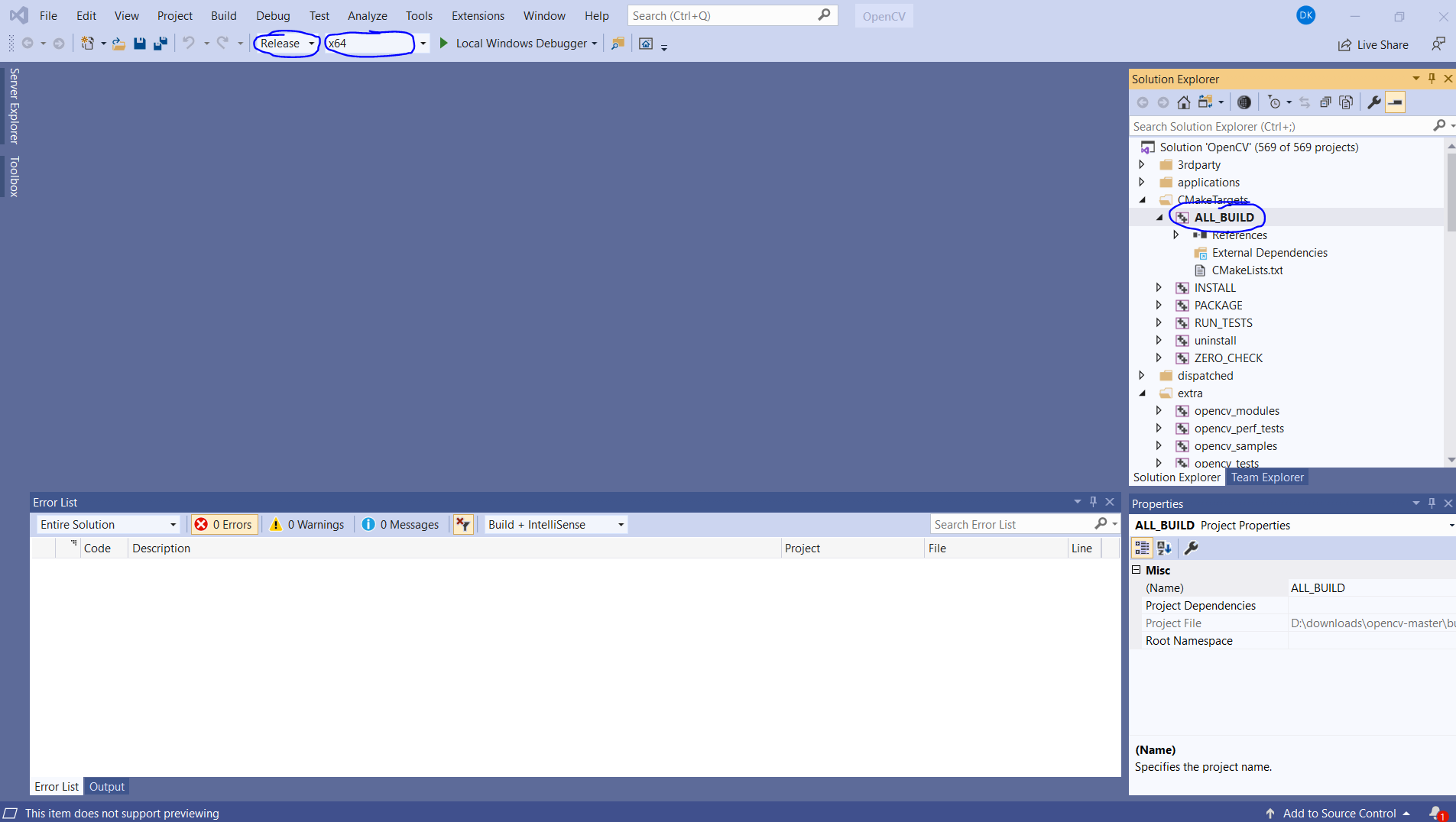


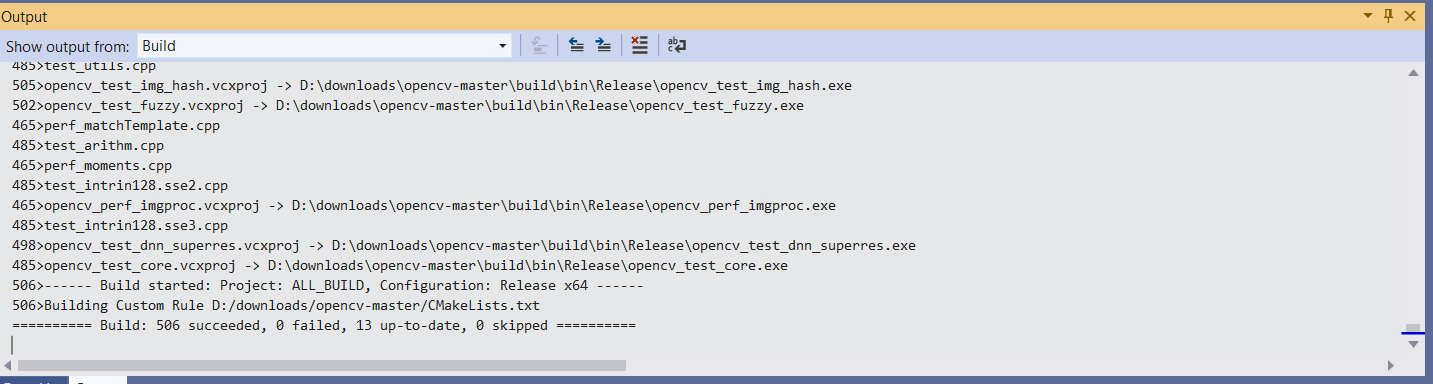
# 

After you hit *generate* you can check in the output log to see if all the correct libraries and extensions were found. You can see here that indeed CUDA was found as well as cuDNN and python. Once all dependencies are verified click on the *open project* button to open the Visual Studio 2019 project. The next step is to compile OpenCV.

# Compile OpenCV

When the Visual Studio project opens you will want to make sure that both *release* and *x64* are selected above. Then right click on ALL\_BUILD and click *build.* This will build the complete OpenCV framework and will take quite some time. Hopefully the build completes successfully and then right click on INSTALL and click *build*. This will install the libraries that were just build into the correct locations. You should see something like the image below that shows that there were no failures. Now your system is all setup to run Visual Studio 2019 programs with OpenCV and CUDA support.



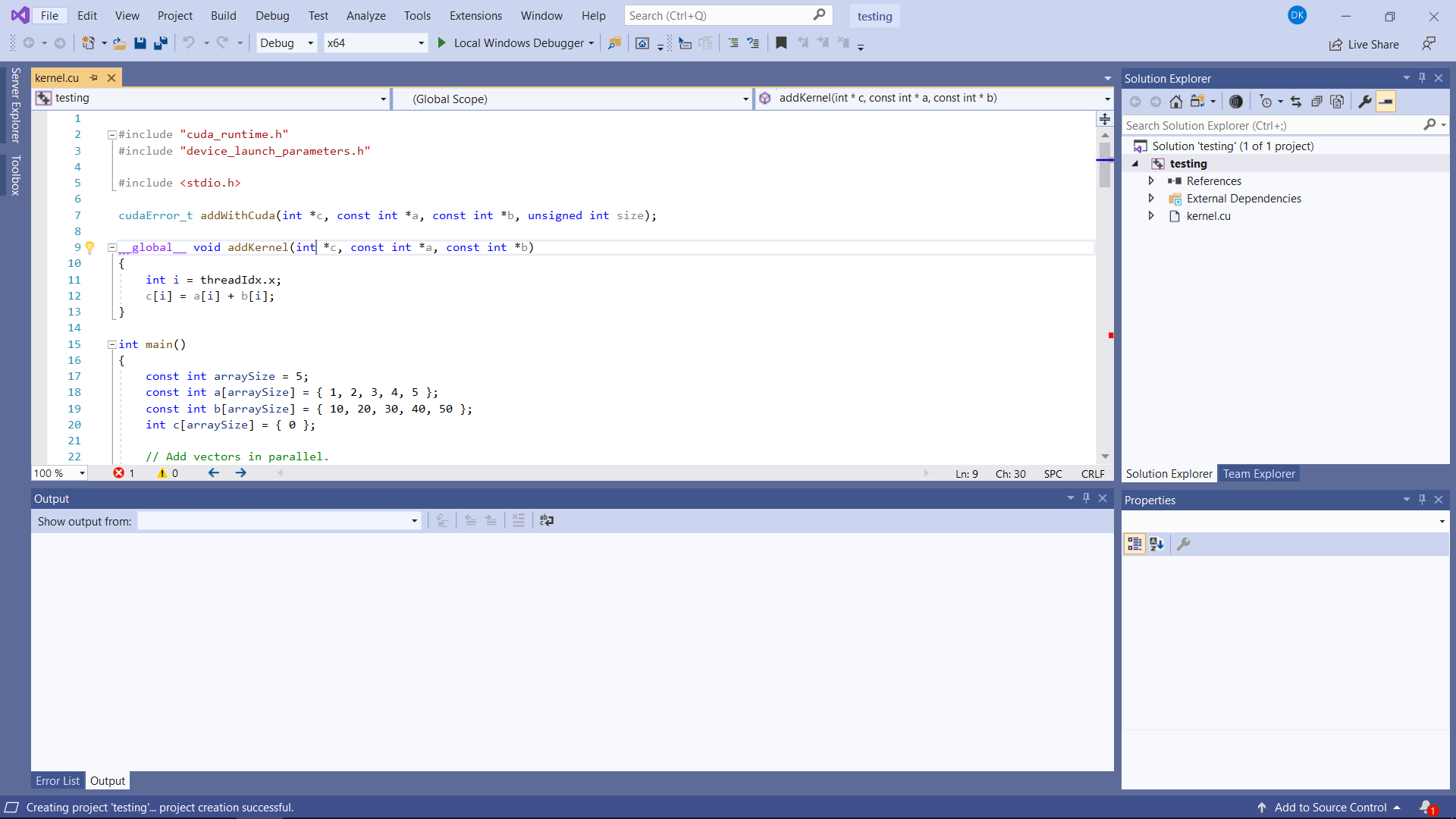


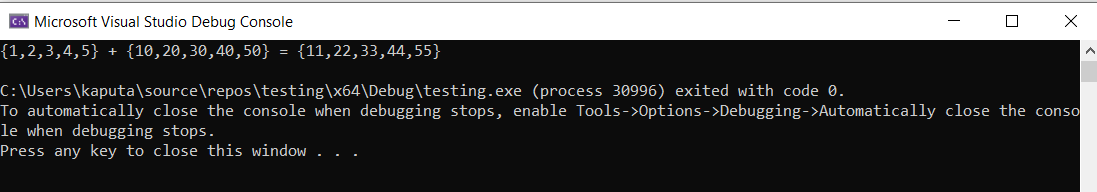
# Verify Visual Studio Works with CUDA

To verify that everything installed correctly it is good to do a ‘hello world’ program. Open up a fresh copy of Visual Studio and click on *Create a new project*. Then scroll all the way down to almost the bottom and select *CUDA 10.2 Runtime*.

|  |  |
| --- | --- |
|  |  |

A default project should open up and look like the image below. Simply click on the green arrow next to *Local Windows Debugger* to compile and run the program. The output should look something like the console capture below.





# Verify Visual Studio Works with OpenCV and CUDA

A little more difficult task is to run Visual Studio 2019 with OpenCV support. As with the previous check, open a fresh version of Visual Studio but this time pay close attention to the directory of the project. Create an empty C++ project as shown below. Note: This entire project can be found via the file aggregate link and is called helloOpenCV-CUDA.

|  |  |
| --- | --- |
|  |  |

Then click on Project => Add new item and name the source file main.cpp

# 

# 

Then copy in the below text into the source file to make sure that we can do a basic hello world. Press F5 to compile and run the code.

|  |
| --- |
| #include <stdio.h>  int main() {  printf("hello world\n");  return 0;  } |

Once you have verified some basic functionality it is time to add in the paths for OpenCV and CUDA. You will have to do the following:

1. Add include paths
2. Add library paths
3. Add dependencies

To add the include paths for OpenCV and CUDA right click on the project from the solution explorer menu and click on properties. Then navigate to C/C++ and click on Additional Include Directories. You will need to add the two below directories. **Make sure to have x64 selected as the platform as it is not the default for some reason**.

|  |
| --- |
| C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.2\include  D:\opencv-cuda\opencv-master\build\install\include |

|  |  |
| --- | --- |
|  |  |

To add the library paths for OpenCV and CUDA right click on the project from the solution explorer menu and click on properties. Then navigate to Linker and click on Additional Library Directories. You will need to add the two below directories.

|  |
| --- |
| C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.2\lib\x64  D:\opencv-cuda\opencv-master\build\install\x64\vc16\lib |

|  |  |
| --- | --- |
|  |  |

The last thing that needs to be done is to add the additional dependencies for OpenCV and CUDA. Navigate to Linker and click on Input. You will need to add the two below libs.

|  |
| --- |
| opencv\_world420.lib  cudnn.lib |

|  |  |
| --- | --- |
|  |  |

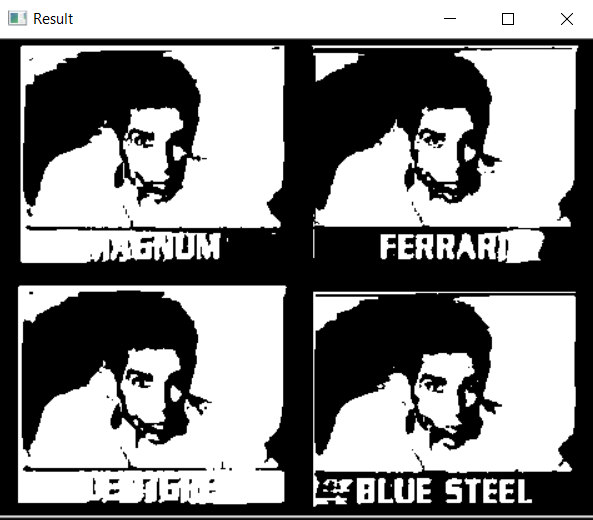
Before the executable can run successfully however the *path* to opencv\_world420.dll must be added to the path environment variable. In my case the path is the string shown below.

**D:\opencv-cuda\opencv-master\build\install\x64\vc16\bin**

After the project has been setup, simply copy the below code into main.cpp and copy oneLook.png into the same directory as main.cpp. oneLook.png can be found in the main directory of the file aggregate mentioned in the top of this document.

|  |
| --- |
| #include <iostream>  #include <opencv2/opencv.hpp>  #include <opencv2/core/cuda.hpp>  using namespace cv;  using namespace std;  int main(int argc, char\* argv[]) {  try {  Mat src\_host = imread("oneLook.png", IMREAD\_GRAYSCALE);  cuda::GpuMat dst, src;  src.upload(src\_host);  cuda::threshold(src, dst, 128.0, 255.0, THRESH\_BINARY);  Mat result\_host;  dst.download(result\_host);  imshow("Result", result\_host);  waitKey();  }  catch (const cv::Exception & ex) {  cout << "Error: " << ex.what() << endl;  }  return 0;  } |

When one runs the above code they should see the below image! Congratulations you have successfully installed OpenCV with CUDA support in Visual Studio 2019 on a Win 10 PC.



# Thanks but no Thanks. Just give me the Binaries

If you don’t want to go through all the above gyrations you can download the pre-compiled binaries that were generated via the above process. They can be found in the binaries folder of the file aggregate link mentioned at the top of this document. You will still need to setup your Visual Studio project and also copy the NVIDIA files to the windows directory but you shouldn’t have to compile anything. Make sure that opencv-master is unzipped to either the same directory as this document uses [d:\opencv-cuda] or make sure to update all paths within this doc to your specific location.

# Links

[**https://jamesbowley.co.uk/accelerating-opencv-4-build-with-cuda-intel-mkl-tbb-and-python-bindings/**](https://jamesbowley.co.uk/accelerating-opencv-4-build-with-cuda-intel-mkl-tbb-and-python-bindings/)

[**https://docs.opencv.org/master/d3/d52/tutorial\_windows\_install.html**](https://docs.opencv.org/master/d3/d52/tutorial_windows_install.html)

[**https://pterneas.com/2018/11/02/opencv-cuda/**](https://pterneas.com/2018/11/02/opencv-cuda/)

[**https://docs.nvidia.com/cuda/cuda-installation-guide-microsoft-windows/index.html**](https://docs.nvidia.com/cuda/cuda-installation-guide-microsoft-windows/index.html)

[**https://docs.microsoft.com/en-us/visualstudio/python/debugging-mixed-mode-c-cpp-python-in-visual-studio?view=vs-2019**](https://docs.microsoft.com/en-us/visualstudio/python/debugging-mixed-mode-c-cpp-python-in-visual-studio?view=vs-2019)

[**https://docs.nvidia.com/deeplearning/sdk/cudnn-install/index.html**](https://docs.nvidia.com/deeplearning/sdk/cudnn-install/index.html)

[**https://jamesbowley.co.uk/accelerating-opencv-with-cuda-streams-in-python/**](https://jamesbowley.co.uk/accelerating-opencv-with-cuda-streams-in-python/)

[**https://jamesbowley.co.uk/accelerating-opencv-4-build-with-cuda-intel-mkl-tbb-and-python-bindings/#cuda\_perfomance**](https://jamesbowley.co.uk/accelerating-opencv-4-build-with-cuda-intel-mkl-tbb-and-python-bindings/#cuda_perfomance)

[**https://cv-tricks.com/how-to/installation-of-opencv-4-1-0-in-windows-10-from-source/**](https://cv-tricks.com/how-to/installation-of-opencv-4-1-0-in-windows-10-from-source/)

[**https://lightbuzz.com/opencv-cuda/**](https://lightbuzz.com/opencv-cuda/)