**Guide for Installing AI & Deep Learning Programs (Linux Ubuntu 16.04 – 17)**

**List of Programs we will be installing in this guide.**

1. **Nvidia Drivers (Latest Version from Nvidia Website)**
2. **Nvidia Cuda 9.1**
3. **Nvidia Digits**
4. **Nvidia cuDNN**
5. **Torch**
6. **Caffe**
7. **PyTorch**
8. **TorchVision**
9. **Tensor Flow (1.7)**
10. **Keras**

**Pre-Installation:**

**Before using this guide make sure you have the following things installed on your system. This guide is applicable for all the Systems running Nvidia Graphics Card/GPU.**

**Make sure you have a fresh copy of Linux Ubuntu 16.04 or greater installed on your system before attempting to use this guide.**

**This guide was developed on a system running Core i7 8700K and GTX 1080Ti but can be used for any Nvidia GPU. I will provide all the steps as easy as possible along with explanations and screen shots.**

**All the commands use in this guide are all single line commands on terminals**

**Nvidia Drivers Installation:**

First, open up a terminal and run this command

**sudo su**

and login as root.

Now after your login its time to update the system. To update your system run this command.

**apt-get update && apt-get upgrade**

After you update the system. Its time to uninstall any older drivers or packages of Nvidia. As you know that Ubuntu by default comes with its own drivers for Nvidia which are on an older version as compare to the current latest drivers. And Cuda 9.1 supports R390 or 390.XX version of drivers.

To uninstall the old drivers please use these command below

**apt-get purge nvidia\***

**apt-get autoremove**

This command will remove your old Nvidia drivers and will allow you to install new drivers. Please note it is not necessary for you to have Nvidia drivers installed of if after running the commands you cannot find any packages for Nvidia its completely fine you can continue ahead.

Now we have the old drivers or installation removed we can go ahead and install the new drivers.

Download the new drivers from Nvidia Official Website. All the supported GPUs are mentioned on the Nvidia drivers page.

<http://www.nvidia.com/download/driverResults.aspx/132530/en-us>

After you download the drivers now its time for installation. The drivers you will download will be a **run** file so before installing we must do a few modifications to that file and install other dependencies for the drivers first.

Run this command.

**apt-get install build-essential gcc-multilib dkms**

The above command will install all the necessary file which we will need for the driver installation.

**Create Blacklist for Nouveau Driver**

Create a file at /etc/modprobe.d/blacklist-nouveau.conf with the following contents:

**blacklist nouveau**

**options nouveau modeset=0**

To create a file open a terminal and type in

**gedit /etc/modprobe.d/blacklist-nouveau.conf**

and copy and paste the above 2 lines and click save.

Now execute **sudo update-initramfs -u** and reboot the computer.

### After you reboot you might notice a few changes on display and resolution. Don’t worry because we will be installing the drivers now.

### Stop lightdm/gdm/kdm

After the computer is rebooted. We need to stop the desktop manager before executing the runfile to install the driver. **lightdm** is the default desktop manager in Ubuntu. If GNOME or KDE desktop environment is used, installed desktop manager will then be **gdm** or **kdm**.

For Ubuntu 16.04, executing

**sudo service lightdm stop** (or use gdm or kdm instead of lightdm)

After you execute the above command the GUI (Graphical User Interface) Will be stopped. Now use this button combination to switch to CLI (Command Line) to install the drivers.

**Ctrl + Alt + F2**

After executing the above command login with your Ubuntu Username and Password which you setup while installing Ubuntu

**Executing the Runfile/Installing Drivers:**

After above batch of preparation, we can eventually start executing the runfile.

Now using the **cd** command navigate to the folder where you downloaded the Drivers. In my case it was in Downloads folder.

**cd ~/Downloads/**

Now type **ls** and it will display you the file which you have downloaded.

The downloaded file will not haver any permissions to run so we need to grant it the executable permission before we can install the drivers. Type in the command below.

**chmod +x (Name of your downloaded file)**

**e.g**

**chmod +x NVIDIA-Linux-x86\_64-390.48.run**

After you have executed the above command we just need to run the installer.

**sudo ./NVIDIA-Linux-x86\_64-390.48.run --dkms -s**

Wait for the driver installation to finish and then reboot your computer by just typing **reboot**  in the terminal.

After the reboot you will see the display back to normal and all the drivers will be installed.

**Check the Installation**

After a successful installation, **nvidia-smi** command will report all your CUDA-capable devices in the system.

### Common Errors and Solutions

1. ERROR: Unable to load the **'nvidia-drm'** kernel module.

* One probable reason is that the system is boot from UEFI but Secure Boot option is turned on in the BIOS setting. Turn it off and the problem will be solved.
* For any other errors please do contact us and let us know.

**Additional Notes:**

**nvidia-smi -pm 1** can enable the persistent mode, which will save some time from loading the driver. It will have significant effect on machines with more than 4 GPUs.

**nvidia-smi -e 0** can disable ECC on TESLA products, which will provide about 1/15 more video memory. Reboot is reqired for taking effect. nvidia-smi -e 1 can be used to enable ECC again.

**nvidia-smi -pl <some power value>** can be used for increasing or decrasing the TDP limit of the GPU. Increasing will encourage higher GPU Boost frequency, but is somehow DANGEROUS and HARMFUL to the GPU. Decreasing will help to same some power, which is useful for machines that does not have enough power supply and will shutdown unintendedly when pull all GPU to their maximum load.

**-i <GPUID>** can be added after above commands to specify individual GPU.

These commands can be added to **/etc/rc.local** for excuting at system boot.

**Nvidia Cuda 9.1 Installation (including Cuda Samples):**

After we have completed the above steps to install the drivers now its time for the Cuda 9.1 Installation. There is a documentation available on Nvidia website as well but it is complicated and does not explain everything well. So, in this guide I will break down the steps and make it as simple as possible.

One of the things which really confused me as well was that Nvidia provides run file or deb file for Cuda installation which comes with the drivers so why did we have to go through the above steps if they already have it in the Cuda installation file. The reason is the Cuda 9.1 requires R390 drivers and the version available in the installation is R387. Don’t know if Nvidia has made a mistake or what but when I attempted to install the drivers using the Cuda Run file it crashed the system and I was unable to get the display or anything working properly.

First of all lets download the CUDA 9.1 From Nvidia Website.

<https://developer.nvidia.com/cuda-downloads?target_os=Linux&target_arch=x86_64&target_distro=Ubuntu&target_version=1604&target_type=runfilelocal>

Please download all the files on that link and we will install them one by one. Make sure you download the RunFiles.

After download all the Run Files. You main Base Installer needs to be extracted first. The reason is that Nvidia Cuda 9.1 base installer comes with 3 files inside one is the Cuda Toolkit another are the samples and the last one are the old drivers. As I mentioned before we cannot use the old drivers because of the Cuda 9.1 support so if we try to install the Run file before extracting it will install all those 3 Run Files inside which can cause a system crash and driver override.

Now after downloading navigate to the installation file folder by **cd** command.

After you are inside the folder which contains the CUDA installation file. Run this command to extract the files.

**chmod +x cuda\_7.5.18\_linux.run**

**./cuda\_7.5.18\_linux.run --extract=$HOME** (It will extract the files inside the root folder **~** use the command

**cd ~**

to navigate inside the root folder and install the CUDA Tool Kit).

Now let’s run the installation file first.

**sudo ./cuda-linux.9.1.85-23083092.run**

Now just follow the instructions on terminal and accept everything. Same goes for installing sample.

**sudo ./cuda-samples.9.1.85-23083092-linux.run**

Please note do not install the drivers as it might crash the system that’s why we extracted the Run File.

**Installing Patches, we downloaded from Cuda Download Page:**

Now we will install the patches which we downloaded. Its pretty simple. All of them are Run File so we will just modify the permission and make them executable and then run them.

**sudo chmod +x (Name of the patch)**

**sudo ./(Name of the patch)**

Install all patches like this and we are ready to setup Cuda.

After the installation is complete we will now setup Cuda for our work environment before we can start using it.

**Setting up the Environmental Variables:**

We will need to setup the environmental variable first so we can access the Cuda Toolkit and commands from anywhere in our terminal.

There are basically 2 ways either you can set them up temporarily or permanent. I will explain both ways.

**Temporary Method for Environment Variables:**

Cope and paste these commands one by one.

**sudo export PATH=/usr/local/cuda/bin${PATH:+:${PATH}}**

**sudo export LD\_LIBRARY\_PATH=/usr/local/cuda/lib64${LD\_LIBRARY\_PATH:+:${LD\_LIBRARY\_PATH}}**

Both the commands are single line commands so in copy & paste if you get a line break in the terminal please type the commands exactly mentioned above. The above method will setup the paths only temporary once you reboot the system it will be gone and you have too set it up again.

Now type **nvcc --help**  if this command works then it means Cuda has successfully installed and working.

**Permanent Method for Environment Variables:**

For permanent we will change and add the Paths inside the PATH file in /etc folder.

On your terminal type

**sudo gedit /etc/environment**

And hit enter. It will open up the file with the PATH. Now copy and paste this line at the end of the pat variable but before the “ marks and put the **:** sign after the /usr/local/games

**/usr/local/cuda/bin**

e.g

**PATH=”/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/usr/local/cuda/bin”**

And save the file. It will permanently set the path until and unless you remove it your self.

Now we are going to copy the library path. Library Path will be copied into the bashrc profile.On you terminal type

**echo "export LD\_LIBRARY\_PATH=$LD\_LIBRARY\_PATH:/usr/local/cuda/lib64" >> ~/.bashrc**

After following the above steps reboot the computer and all the paths will be setup for you. Now type **nvcc --help**  if this command works then it means Cuda has successfully installed and working.

You might need to install additional libraries while working with Image Processing.

**sudo apt-get install g++ freeglut3-dev build-essential libx11-dev libxmu-dev libxi-dev libglu1-mesa libglu1-mesa-dev**

After you install additional libraries a symbolic link can be broken. So, what you need to do is reinstall the driver which you downloaded in the first step. But before that reboot the computer and press **CTRL + ALT + F2**  to get a terminal screen and stop the lightdm service

**sudo service lightdm stop** (or use gdm or kdm instead of lightdm)

After that navigate to where you downloaded the Run File and reinstall drivers.

**sudo ./NVIDIA-Linux-x86\_64-390.48.run --dkms -s**

Wait for the driver installation to finish and then reboot your computer by just typing **reboot**  in the terminal.

After the reboot you will see the display back to normal and all the drivers will be installed. Nad the broken links will be fixed.

### [7.2.1. Install Persistence Daemon](http://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html" \l "install-persistenced)

The NVIDIA Persistence Daemon can be started as the root user by running:

**/usr/bin/nvidia-persistenced –verbose**

Add it to the /etc/rc.local to make it run on every time when the system boots up.

**[7.1.2. POWER9 Setup](http://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html" \l "power9-setup) For Tesla GPUs only. Not for other Nvidia GPUs**

Because of the addition of new features specific to the NVIDIA POWER9 CUDA driver, there are some additional setup requirements in order for the driver to function properly. These additional steps are not handled by the installation of CUDA packages, and failure to ensure these extra requirements are met will result in a non-functional CUDA driver installation.

There are two changes that need to be made manually after installing the NVIDIA CUDA driver to ensure proper operation:

1. Create and enable a systemd service file or init script that runs the NVsIDIA Persistence Daemon as the first NVIDIA software during or at the end of the boot process. The following service file example is sufficient for most installations:

[Unit]

Description=NVIDIA Persistence Daemon

Wants=syslog.target

[Service]

Type=forking

PIDFile=/var/run/nvidia-persistenced/nvidia-persistenced.pid

Restart=always

ExecStart=/usr/bin/nvidia-persistenced --verbose

ExecStopPost=/bin/rm -rf /var/run/nvidia-persistenced

[Install]

WantedBy=multi-user.target

Copy the above text into the following file:

/usr/lib/systemd/system/nvidia-persistenced.service

And run the following command:

**$** sudo systemctl enable nvidia-persistenced

1. Disable a udev rule installed by default in some Linux distributions that cause hot-pluggable memory to be automatically onlined when it is physically probed. This behavior prevents NVIDIA software from bringing NVIDIA device memory online with non-default settings. This udev rule must be disabled in order for the NVIDIA CUDA driver to function properly on POWER9 systems.

On RedHat Enterprise Linux 7, this rule can be found in:

/lib/udev/rules.d/40-redhat.rules

On Ubuntu 17.04, this rule can be found in:

/lib/udev/rules.d/40-vm-hotadd.rules

The rule generally takes a form where it detects the addition of a memory block and changes the 'state' attribute to online. For example, in RHEL7, the rule looks like this:

SUBSYSTEM=="memory", ACTION=="add", PROGRAM="/bin/uname -p", RESULT!="s390\*", ATTR{state}=="offline", ATTR{state}="online"

This rule must be commented out, removed, or changed so that it does not apply to POWER9 NVIDIA systems.

You will need to reboot the system to initialize the above changes.

**Nvidia cuDNN Installation:**

Download cuDNN from Nvidia Official Website. Note: You will need to create an account first and then you will be able to download the cuDNN. Download the latest version. There will be multiple downloads available. Download the normal version as the Power8/Power9 is for Tesla GPUs only.

cuDNN v7.1.2 Library for Linux

cuDNN v7.1.2 for Linux (Power8/Power9) – Onl;y for Tesla GPUs

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

There will also be deb packages available as well, but we will do the installation manually as we did while installing Cuda.

After downloading the **tar** file extract it to the folder.

**sudo -xzvf cudnn-9.1-linux-x64-v7.1.tgz**

After extracting this will create a folder named cuda inside the directory where you extracted the file. Now you will need to copy the libraries into the cuda folder.

Now Copy the following files into the CUDA Toolkit directory.

**sudo cp cuda/include/cudnn.h /usr/local/cuda/include/**

**sudo cp cuda/lib64/libcudnn\* /usr/local/cuda/lib64/**

**sudo chmod a+r /usr/local/cuda/include/cudnn.h**

**sudo chmod a+r** **/usr/local/cuda/lib64/libcudnn\***

After this your cuDNN installation is complete. It’s a pretty simple installation. Just need to have a little understanding of linux.

**Nvidia Digits Installation:**

In this tutorial we will install Nvidia Digits. But before installing nvidia Digist we need to install 2 other programs

1. **Docker**
2. **Nvidia-Docker (For Digits to run smoothly)**
3. **Caffe**
4. **TensorFlow**
5. **Torch7**

**Docker Installation:**

First, we will install Nvidia Docker. If you have freshly installed Ubuntu and have been following the guide you will not need to remove anything as Docker is not by default by Ubuntu. If you do have Docker installed no need to worry just type

**sudo apt-get remove docker docker-engine docker-io**

This will uninstall older version of Docker, so you can continue to install the newer version.

First of all we will update our repositories to do that type

**sudo apt-get update**

Now we have to allow apt to use repositories over HTTPS as well:

**sudo apt-get install apt-transport-https ca-certificates curl software-properties-common**

Now we will add Docker’s official GPG Key:

**sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -**

Now verify that you now have the key with the fingerprint 9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88, by searching for the last 8 characters of the fingerprint.

**sudo apt-key fingerprint 0EBFCD88**

Now we will set up the stable repository.

**sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable"**

After this we will perform a **apt-get update** to make sure all the repos are updated and the new repos are added as well.

Now top install docker

**sudo apt-get install docker-ce**

After this docker will be installed.

Now just run to verify docker installation.

**sudo docker run hello-world**

**Installing Nvidia Docker:**

Add the Nvidia Docker Repository

**sudo curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | sudo apt-key add -**

After that lets install docker

**curl -s -L https://nvidia.github.io/nvidia-docker/ubuntu16.04/amd64/nvidia-docker.list | sudo tee /etc/apt/sources.list.d/nvidia-docker.list**

**(Note this is for Ubuntu 16.04 for 17 or above change the link or visit official docker installation by Nvidia)**

**sudo apt-get update**

**sudo apt-get install -y nvidia-docker2**

**sudo pkill -SIGHUP dockerd**

Now we will create the image which will be used by the Docker

**docker run --runtime=nvidia --rm nvidia/cuda nvidia-smi**

It will now download the image so wait a while once its done it will show you the image on the terminal.

**Caffe Installation:**

**Before installing Caffe we need to install Protobuf**

These Deb packages must be installed to build Protobuf 3

**sudo apt-get install autoconf automake libtool curl make g++ git python-dev python-setuptools unzip**

DIGITS is currently compatiable with Protobuf 3.2.x

# example location - can be customized

**sudo export PROTOBUF\_ROOT=~/protobuf**

**sudo git clone https://github.com/google/protobuf.git $PROTOBUF\_ROOT -b '3.2.x'**

**sudo cd $PROTOBUF\_ROOT**

**sudo ./autogen.sh**

**sudo ./configure**

**sudo make "-j$(nproc)"**

**sudo make install**

**sudo ldconfig**

**sudo cd python**

**sudo python setup.py install --cpp\_implementation**

After this we will now install Caffe

First, we need to install these dependencies for Caffe

**sudo apt-get install --no-install-recommends git graphviz python-dev python-flask python-flaskext.wtf python-gevent python-h5py python-numpy python-pil python-pip python-scipy python-tk**

**sudo apt-get install --no-install-recommends build-essential cmake git gfortran libatlas-base-dev libboost-filesystem-dev libboost-python-dev libboost-system-dev libboost-thread-dev libgflags-dev libgoogle-glog-dev libhdf5-serial-dev libleveldb-dev liblmdb-dev libopencv-dev libsnappy-dev python-all-dev python-dev python-h5py python-matplotlib python-numpy python-opencv python-pil python-pip python-pydot python-scipy python-skimage python-sklearn**

**sudo apt-get install libprotobuf-dev libleveldb-dev libsnappy-dev libopencv-dev libhdf5-serial-dev protobuf-compiler libturbojpeg**

**sudo apt-get install --no-install-recommends libboost-all-dev**

**sudo apt-get install libatlas-base-dev**

**sudo ln -s /usr/lib/x86\_64-linux-gnu/libturbojpeg.so.0.1.0 /usr/lib/x86\_64-linux-gnu/libturbojpeg.so**

**sudo apt-cache search openblas**

**sudo apt-get install libopenblas-dev**

**sudo export CAFFE\_ROOT=~/caffe**

**sudo git clone https://github.com/NVIDIA/caffe.git $CAFFE\_ROOT -b 'caffe-0.17'**

**sudo pip install -r $CAFFE\_ROOT/python/requirements.txt**

**sudo cat $CAFFE\_ROOT/python/requirements.txt | xargs -n1 sudo pip install**

**sudo cd $CAFFE\_ROOT/**

**sudo mkdir build**

**sudo cd build/**

**sudo cmake ..**

**sudo make -j"$(nproc)"**

**sudo make install**

**TensorFlow Installation:**

Now we will install Tensor Flow. But before that we need to install a few dependencies to build tensorflow from source.

1. Bazel
2. TensorFlow Python Dependencies

**Bazel:**

To install first we need to install openjdk-8 as bezel support openjdk version 8.

**sudo apt-get install openjdk-8-jdk**

**echo "deb [arch=amd64] http://storage.googleapis.com/bazel-apt stable jdk1.8" | sudo tee /etc/apt/sources.list.d/bazel.list**

**curl https://bazel.build/bazel-release.pub.gpg | sudo apt-key add –**

**sudo apt-get update && sudo apt-get install bazel**

**sudo apt-get install python-numpy python-dev python-pip python-wheel**

**sudo apt-get install python3-numpy python3-dev python3-pip python3-wheel**

**sudo export LD\_LIBRARY\_PATH=$LD\_LIBRARY\_PATH:/usr/local/cuda/extras/CUPTI/lib64**

**TensorFlow:**

**sudo git clone https://github.com/tensorflow/tensorflow**

**./configure**

**apt-get install librdmacm-dev**

**apt-get install libibverbs-dev**

**bazel build --config=opt --config=cuda //tensorflow/tools/pip\_package:build\_pip\_package --verbose\_failures**

**bazel-bin/tensorflow/tools/pip\_package/build\_pip\_package /tensorflow\_pkg**

**sudo pip install /tensorflow\_pkg/tensorflow-1.7.0-cp27-cp27mu-linux\_x86\_64.whl**

**Validate your installation**

**Validate your TensorFlow installation by doing the following:**

**Start a terminal.**

**Change directory (cd) to any directory on your system other than the tensorflow subdirectory from which you invoked the configure command.**

**Invoke python:**

**$ python**

**Enter the following short program inside the python interactive shell:**

**# Python**

**import tensorflow as tf**

**hello = tf.constant('Hello, TensorFlow!')**

**sess = tf.Session()**

**print(sess.run(hello))**

**If the system outputs the following, then you are ready to begin writing TensorFlow programs:**

**Hello, TensorFlow!**

**Torch7 Installation:**

Torch can be installed to your home folder in ~/torch by running these three commands:

# in a terminal, run the commands WITHOUT sudo

**git clone https://github.com/torch/distro.git ~/torch --recursive**

**cd ~/torch; bash install-deps;**

**./install.sh**

The first script installs the basic package dependencies that LuaJIT and Torch require. The second script installs LuaJIT, LuaRocks, and then uses LuaRocks (the lua package manager) to install core packages like torch, nn and paths, as well as a few other packages.

The script adds torch to your PATH variable. You just have to source it once to refresh your env variables. The installation script will detect what is your current shell and modify the path in the correct configuration file.

**# On Linux with bash**

**source ~/.bashrc**

**# On Linux with zsh**

**source ~/.zshrc**

**# run luarocks WITHOUT sudo**

**$ luarocks install image**

**$ luarocks list**

**Once installed you can run torch with the command th from you prompt!**

**The easiest way to learn and experiment with Torch is by starting an interactive session (also known as the torch read-eval-print loop or TREPL):**

**$ th**

**\_\_\_\_\_\_ \_\_ | Torch7**

**/\_ \_\_/\_\_ \_\_\_\_\_\_\_\_/ / | Scientific computing for Lua.**

**/ / / \_ \/ \_\_/ \_\_/ \_ \ |**

**/\_/ \\_\_\_/\_/ \\_\_/\_//\_/ | https://github.com/torch**

**| http://torch.ch**

**th> torch.Tensor{1,2,3}**

**1**

**2**

**3**

**[torch.DoubleTensor of dimension 3]**

**th>**

**Installing Keras, Pytorch & TorchVision:**

**sudo pip install keras**

**For Pytorch & TorchVision Please visit** [**http://pytorch.org/**](http://pytorch.org/) **and download the appropriate version. Its simple to ionstall these programs.**

**Installing Nvidia Digits:**

**sudo apt-get install --no-install-recommends git graphviz python-dev python-flask python-flaskext.wtf python-gevent python-h5py python-numpy python-pil python-pip python-scipy python-tk**

**DIGITS\_ROOT=~/digits**

**git clone https://github.com/NVIDIA/DIGITS.git $DIGITS\_ROOT**

**sudo pip install -r $DIGITS\_ROOT/requirements.txt**

**pip install -e $DIGITS\_ROOT**

**cd /usr/local/bin/**

**./digits-devserver**

**Make sure to add these path in your bashrc profile. Or you can run these commands one by one so add them** **export CAFFE\_ROOT=~/caffe**

**export TORCH\_ROOT=~/torch**

**export PATH=/usr/local/cuda/bin:${PATH}**

**export LD\_LIBRARY\_PATH=/usr/local/cuda/lib64:$LD\_LIBRARY\_PATH**

**export LD\_LIBRARY\_PATH=/usr/local/cuda/extras/CUPTI/lib64:${LD\_LIBRARY\_PATH}**

**. /root/torch/install/bin/torch-activate manually after a reboot.**

**Thank you for following my guide. These are the step by step instructions to set up your own AI &Deep Learning build on your computer. The OS used in this entire build is Ubuntu 16.04 LTS along with a Core i7 8700K & GTX 1080Ti. This guide is only for Nvidia GPUs. Please make sure that you follow all the steps from top to bottom. If you face any trouble in any of the steps please do let me know.**

**You can send me an email or contact me VIA LinkedIn or Facebook.**

**I will also be creating a Customized Distribution for this so you can just download and run everything out of the box. Currently testing the Distribution as soon as its complete will make it public to download for free.**

**Email:** [**ammarshah1111@gmail.com**](mailto:ammarshah1111@gmail.com)

**LinkedIn:** [**https://www.linkedin.com/in/ammar-ali-shah-b6b00469/**](https://www.linkedin.com/in/ammar-ali-shah-b6b00469/)

**Facebook:** [**https://www.facebook.com/ProgrammerShah**](https://www.facebook.com/ProgrammerShah)