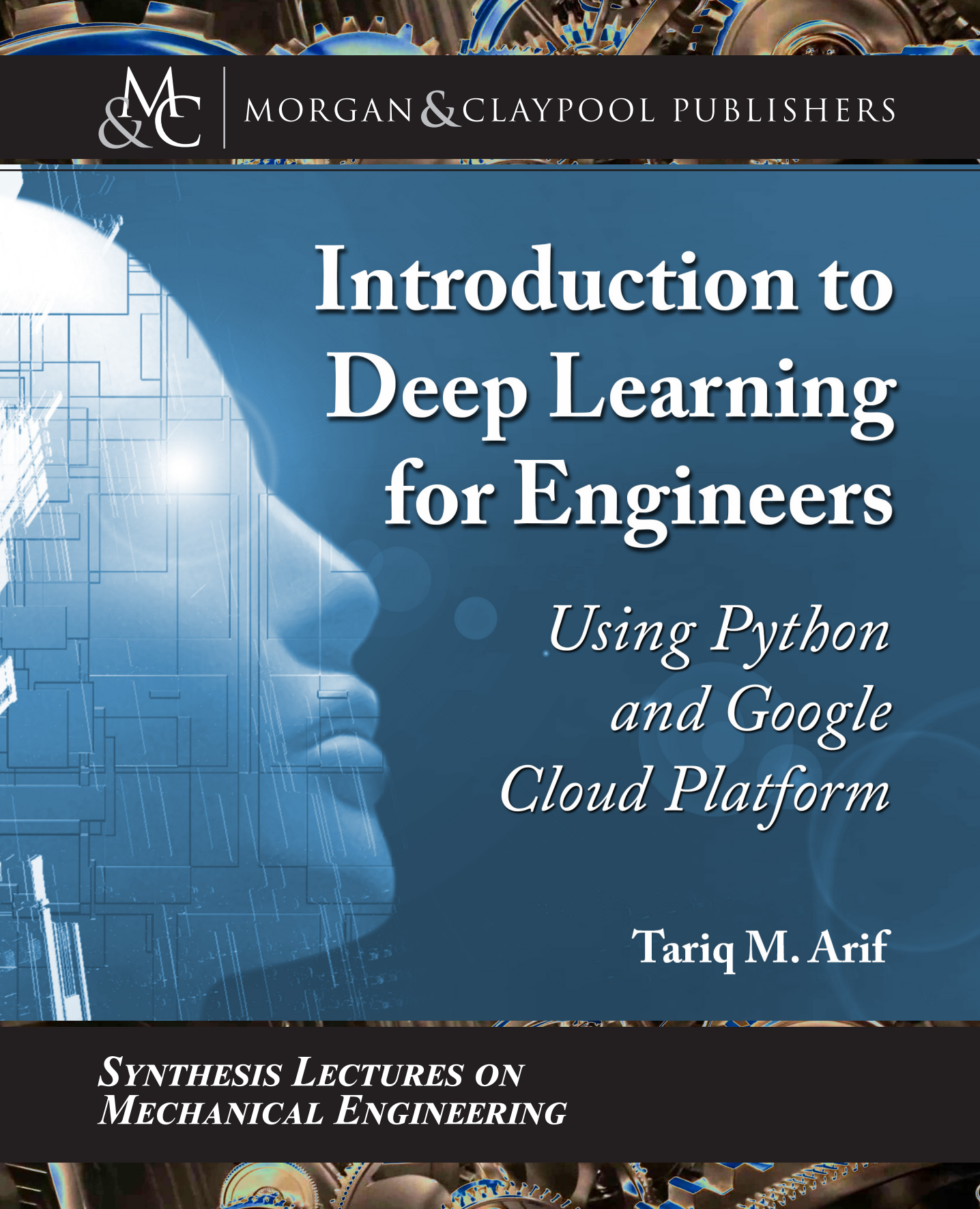
MORGAN & CLAYPOOL PUBLISHERS
Introduction to Deep Learning for Engineers
Using Python and Google Cloud Platform
Tariq M. Arif
Synthesis Lectures on Mechanical Engineering

Introduction to Deep Learning for Engineers

Using Python and Google Cloud Platform

Synthesis Lectures on Mechanical Engineering

Synthesis Lectures on Mechanical Engineering series publishes 60–150 page publications pertaining to this diverse discipline of mechanical engineering. The series presents Lectures written for an audience of researchers, industry engineers, undergraduate and graduate students.
Additional Synthesis series will be developed covering key areas within mechanical engineering.

Introduction to Deep Learning for Engineers: Using Python and Google Cloud Platform
Tariq M. Arif
2020

Towards Analytical Chaotic Evolutions in Brusselators
Albert C.J. Luo and Siyu Guo
2020

Modeling and Simulation of Nanofluid Flow Problems
Snehashi Chakraverty and Uddhaba Biswal
2020

Modeling and Simulation of Mechatronic Systems using Simscape
Shuvra Das
2020

Automatic Flight Control Systems
Mohammad Sadraey
2020

Bifurcation Dynamics of a Damped Parametric Pendulum
Yu Guo and Albert C.J. Luo
2019

Reliability-Based Mechanical Design, Volume 2: Component under Cyclic Load and Dimension Design with Required Reliability
Xiaobin Le
2019

Reliability-Based Mechanical Design, Volume 1: Component under Static Load
Xiaobin Le
2019

Solving Practical Engineering Mechanics Problems: Advanced Kinetics
Sayavur I. Bakhtiyarov
2019

Natural Corrosion Inhibitors
Shima Ghanavati Nasab, Mehdi Javaheran Yazd, Abolfazl Semnani, Homa Kahkesh, Navid Rabiee, Mohammad Rabiee, and Mojtaba Bagherzadeh
2019

Fractional Calculus with its Applications in Engineering and Technology
Yi Yang and Haiyan Henry Zhang
2019

Essential Engineering Thermodynamics: A Student's Guide
Yumin Zhang
2018

Engineering Dynamics
Cho W.S. To
2018

Solving Practical Engineering Problems in Engineering Mechanics: Dynamics
Sayavur Bakhtiyarov
2018

Solving Practical Engineering Mechanics Problems: Kinematics
Sayavur I. Bakhtiyarov
2018

C Programming and Numerical Analysis: An Introduction
Seiichi Nomura
2018

Mathematical Magnetohydrodynamics
Nikolas Xiros
2018

Design Engineering Journey
Ramana M. Pidaparti
2018

Introduction to Kinematics and Dynamics of Machinery
Cho W. S. To
2017

Microcontroller Education: Do it Yourself, Reinvent the Wheel, Code to Learn
Dimosthenis E. Bolanakis
2017

Solving Practical Engineering Mechanics Problems: Statics
Sayavur I. Bakhtiyarov
2017

Unmanned Aircraft Design: A Review of Fundamentals
Mohammad Sadraey
2017

Introduction to Refrigeration and Air Conditioning Systems: Theory and Applications
Allan Kirkpatrick
2017

Resistance Spot Welding: Fundamentals and Applications for the Automotive Industry
Menachem Kimchi and David H. Phillips
2017

MEMS Barometers Toward Vertical Position Detection: Background Theory, System Prototyping, and Measurement Analysis
Dimosthenis E. Bolanakis
2017

Engineering Finite Element Analysis
Ramana M. Pidaparti
2017

Introduction to Deep Learning for Engineers: Using Python and Google Cloud Platform

Tariq M. Arif

www.morganclaypool.com

ISBN: 9781681739137 paperback
ISBN: 9781681739144 ebook
ISBN: 9781681739151 hardcover

DOI 10.2200/S01029ED1V01Y202007MEC028

A Publication in the Morgan & Claypool Publishers series
SYNTHESIS LECTURES ON MECHANICAL ENGINEERING

Lecture #28
Series ISSN
Print 2573-3168 Electronic 2573-3176

Introduction to Deep Learning for Engineers

Using Python and Google Cloud Platform

Tariq M. Arif
Weber State University

SYNTHESIS LECTURES ON MECHANICAL ENGINEERING #28

MORGAN & CLAYPOOL PUBLISHERS

ABSTRACT

This book provides a short introduction and easy-to-follow implementation steps of deep learning using Google Cloud Platform. It also includes a practical case study that highlights the utilization of Python and related libraries for running a pre-trained deep learning model.

In recent years, deep learning-based modeling approaches have been used in a wide variety of engineering domains, such as autonomous cars, intelligent robotics, computer vision, natural language processing, and bioinformatics. Also, numerous real-world engineering applications utilize an existing pre-trained deep learning model that has already been developed and optimized for a related task. However, incorporating a deep learning model in a research project is quite challenging, especially for someone who doesn't have related machine learning and cloud computing knowledge. Keeping that in mind, this book is intended to be a short introduction of deep learning basics through the example of a practical implementation case.

The audience of this short book is undergraduate engineering students who wish to explore deep learning models in their class project or senior design project without having a full journey through the machine learning theories. The case study part at the end also provides a cost-effective and step-by-step approach that can be replicated by others easily.

KEYWORDS

Google Cloud Platform (GCP), Python, PyTorch, artificial neural network, machine learning, deep learning, transfer learning, pre-trained model, convolutional neural network, pooling layers, EfficientNet

Contents

Preface

My goal in writing this book is to motivate engineering students to learn deep learning applications. As a newly evolved area, deep learning models and their implementation methods are rapidly changing. This nature of continuous development provides an extra layer of challenges for newcomers, especially those who are not in the field of data science or machine learning. I found it exciting that many engineering projects can be optimized and improved significantly by implementing a deep learning model that is already tuned for a similar application. Currently, there are numerous excellent books and online resources available on deep learning to explain theories and applications. But I always felt the need for a short book that shows all the necessary techniques to utilize deep learning models in real-life projects using cloud computing. So, my primary motivation in writing this manuscript is to give a whole picture of the application process.

One of the challenges I faced while writing is to squeeze the introductory chapters that cover fundamental theories. I would suggest readers relate the case study part with fundamental theories and focus on running the codes provided in this book by setting up a GCP account. Although this book covers only a small facet of deep learning implementation, I hope it will inspire engineering students to learn other different variations that can be related to their projects.

Tariq M. Arif
July 2020

Acknowledgments

I would like to thank Mr. Paul Petralia, Dr. C.L. Tondo, Ms. Melanie Carlson, and all of the members at Morgan & Claypool Publishers who are involved in improving and editing this manuscript at different stages. I would also like to thank Mr. Adilur Rahim for his expert inputs on deep learning and motivating me to work in this field.

I must thank my wife, Mahbuba Sultana, for being tremendously supportive, and I am also grateful to my daughter, Nuha Arif, who sometimes decided not to distract me while I was working on this manuscript.

Tariq M. Arif
July 2020

CHAPTER 1

Introduction: Python and Array Operations

1.1 INTRODUCTION

Python is a popular object-oriented programming language that was initially created by Guido van Roussum in 1990. Later on, many developers and programmers contributed to its growth, and its popularity has been escalated in recent years in many different domains, including machine learning [1]. Python also has many vibrant online communities and active support forums. Among the popular programming forums, the https://stackoverflow.com/ and https://python-forum.io/ sites have a substantial amount of useful information for beginners. One of the benefits of using Python is that it provides numerous ready-to-use robust libraries that are actively maintained by developers through the GitHub page (https://github.com/). Many of its libraries can be implemented effectively for general-purpose programming, scientific computing, data visualization, and machine learning applications. Beside freely available libraries, there are numerous other reasons why Python become a popular programming language, not only for machine learning or artificial intelligence but also for web, game development, and business applications. The Python hype is likely to grow for many more years to come, and recently, many top U.S. universities have switched to Python for teaching introductory programming classes [2].

Deep learning is a new and popular platform that is widely used in many different predictive problems related to artificial intelligence. Many scientists and researchers use Python programming language and its libraries to utilize deep learning models. Training a deep learning model requires an extensive amount of data processing capabilities by GPU-based hardware, which is relatively expensive. Nowadays, there are cloud computing platforms available such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP), and these platforms can be used for training a data-intensive deep learning model. In this book, we have shown the steps required to run a deep learning model using the google cloud platform in a cost-effective way. Since the basic Python programming is required to follow this text, the introductory chapters (Chapters 1 and 2) are included to provide relevant Python programming skills and library descriptions. For running deep learning model, a widely used library known as NumPy (Numerical Python) and a tensor array library (PyTorch) are used.

Next, Chapters 3–5 provide basic introductions related to the artificial neural network, deep learning, and transfer learning frameworks. Chapter 6 illustrates the essential tips and

tricks to set up the Google Cloud Platform's GPU engine and the PyTorch library. Finally, Chapter 7 presents a case study for the actual implementation of a deep learning model.

Although introductory backgrounds are provided, anyone who is completely new to Python should read some basics of Python data structure and string manipulations before starting Chapter 7. Numerous materials are freely available online to learn the basics of Python programming. A good starting point is the official Python tutorials available at https://docs.python.org/3/tutorial/.

In the following sections, we are going to show the steps to install Anaconda and run Jupyter Notebook. These steps will be useful to follow the case study presented at the end. Later in this chapter, we will show a few simple array manipulations using Python and NumPy library.

1.2 ANACONDA INSTALLATION

We are going to use an open-source distribution for Python, known as Anaconda, which includes a lot of useful scientific packages for machine learning applications and data processing. One of the benefits of using Anaconda is that it comes with hundreds of libraries that are not required to install separately for our workflows such as NumPy, Matplotlib, pandas, SciPy, Bokeh, and many more. To download Anaconda visit https://www.anaconda.com/products/individual. Once the download option has been selected, there will be options for selecting the installer associated with different operating platforms (e.g., Windows/ MacOS/ Linux). While installing the Anaconda, make sure to check on both of the advanced options, as shown in Fig. 1.1.

If Python had already been installed on your computer, this selection would help to create the correct PATH variable needed for the installation.

1.3 USING JUPYTER NOTEBOOK

After Anaconda installation Anaconda Navigator can be opened from the program search menu, which provides several interactive computing environments, including the Jupyter Notebook. We can open Jupyter Notebook from Anaconda, or we may search in the search menu for "Jupyter Notebook (Anaconda3)" and select from there.

The Jupyter Notebook (shown in Fig. 1.2) allows us to create live Python code and iterate data effectively for machine learning applications. This notebook editor is tremendously popular and became a de facto standard among data scientists. One study shows that in September 2018, the code-sharing site GitHub counted more than 2.5 million public Jupyter Notebooks [3].

1.4 ARRAY COMPUTING USING NUMPY

The NumPy (Numerical Python) library can be used to handle a massive amount of array-based dataset. It was first introduced in 2006, and due to its high performance in handling big-data for deep learning, many other supporting libraries such as Pandas and SciPy (Scientific Python)

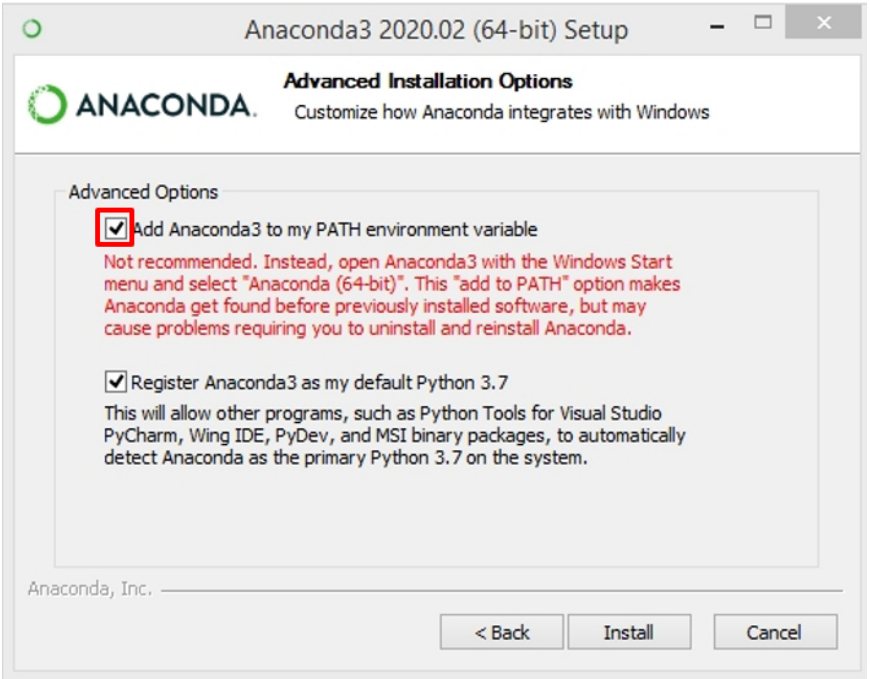

Figure 1.1: Select "Add Anaconda3 to my PATH environment variable" during installation.

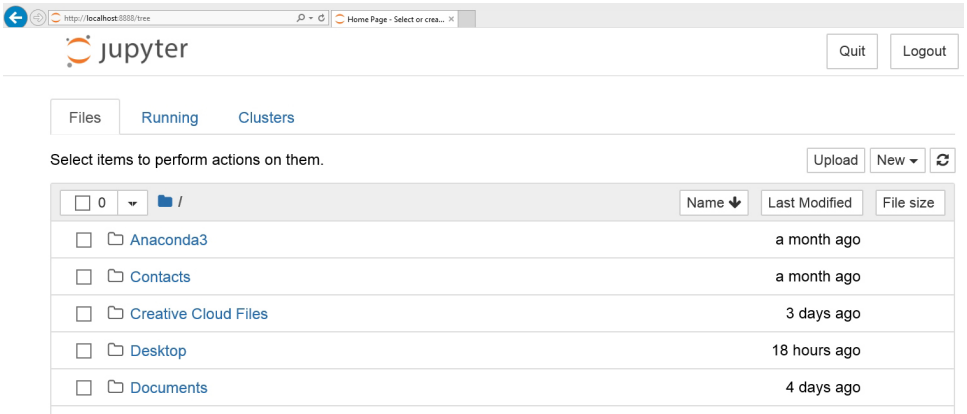

Figure 1.2: Computing environment by Jupyter Notebook.

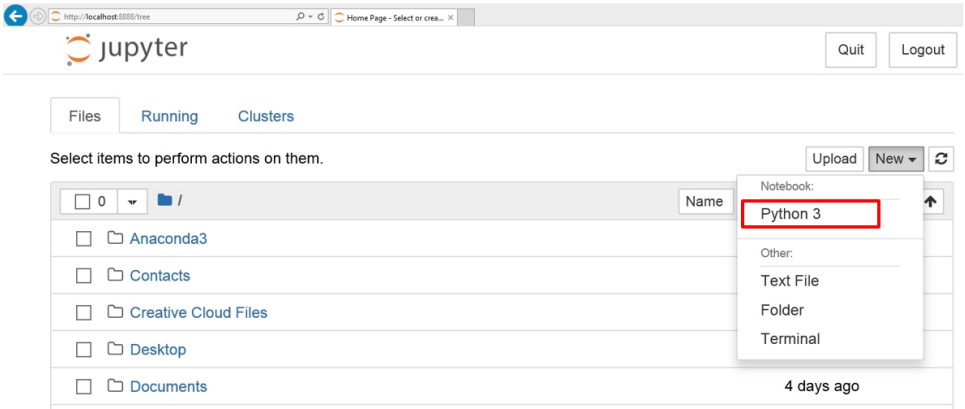

Figure 1.3: Go to new → Python3 to create a new programming script.

Figure 1.4: Import `numpy` library as `np`.

are created based on NumPy library. The NumPy module can be imported as "np" (different keywords can be chosen), and after importing, it can be used to call any array by typing "np."

1.4.1 IMPORTING NUMPY AND BASIC COMMANDS

Let us use some basic functionalities of NumPy. First, open Jupyter Notebook (Anaconda3) from the search menu and create a new "Python 3" file (Fig. 1.3).

Type the following command to import the NumPy library as "np" and press "Shif+Enter" key to execute the line (Fig. 1.4).

Now let's say we have a list of data, `A` = [1, 2, 3, 4, 5, 6, 7, 8, 9], and we would like to convert it to a 3 by 3 matrix. We may assign it to another array variable `B`. Note that in Fig. 1.5, the output "`type`" of `A` is a list, but the output "`type`" of `B` is a `numpy.ndarray`.

Now, we can use the "`reshape`" command to make a 3 by 3 matrix from `B`, but it will not change the original dimension of `B`, as shown in Fig. 1.6. If we want to change the original dimension, we will need to assign it to another variable, or `B` itself.

In Fig. 1.7, array `B` is assigned again using the `reshape` command. Here, the original data shape of `B` also changed to 3 by 3.

```
In [2]: A = [1,2,3,4,5,6,7,8,9]

In [3]: type(A)
Out[3]: list

In [4]: B = np.array(A)

In [5]: type(B)
Out[5]: numpy.ndarray
```

Figure 1.5: Using `numpy` for converting python list to `array` `type`.

```
In [6]: B.reshape(3,3)
Out[6]: array([[1, 2, 3],
               [4, 5, 6],
               [7, 8, 9]])

In [7]: print(B)
        [1 2 3 4 5 6 7 8 9]
```

Figure 1.6: Reshaping `array` `B` without changing its data.

```
In [8]: B = B.reshape(3,3)

In [9]: print(B)
        [[1 2 3]
         [4 5 6]
         [7 8 9]]
```

Figure 1.7: Use `reshape` to change the original array dimension.

1.4.2 CHANGING ARRAY ELEMENTS

Changing element values of an array can be useful when we deal with a multi-dimensional large dataset. But here, we are providing an example on the small array, `B` (Fig. 1.7). Let's say we want to change all the elements of row two to zeros.

Note that a Python list, as well as NumPy array, always starts from 0, i.e., in the 3 by 3 matrix `B`, we have rows from 0 to 2 ($0, 1, 2 \rightarrow 3$ rows), and columns from 0 to 2 ($0, 1, 2 \rightarrow 3$ columns). So when we try to access row 2, we will use `B[1, :]`, as shown in Fig. 1.8. Here, `":"` denotes that we want to reassign all the values (row 1 and column all).

Now, let's try to update only one element located at row 3 and column 3, from 9 to 999. To access this location in NumPy, we will use `[2, 2]`, as shown in Fig. 1.9.

```
In [10]: B[1,:] = 0

In [11]: print(B)

[[1 2 3]
 [0 0 0]
 [7 8 9]]
```

Figure 1.8: Reassigning row values inside of an array.

```
In [12]: B[2,2] = 999

In [13]: B

Out[13]: array([[  1,   2,   3],
                [  0,   0,   0],
                [  7,   8, 999]])
```

Figure 1.9: Updating only one element of an array located at `[2,2]`.

```
In [14]: C = np.zeros(10000).reshape(100,100)

In [15]: C

Out[15]: array([[0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.],
                ...,
                [0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.]])
```

Figure 1.10: Creating a 100 by 100 array of zeros using NumPy.

Note that we can also use "`B`" in the command line instead of "`print(B)`", and press "Shift+Enter" from the keyboard to see the array output.

1.4.3 CREATING ARRAY WITH SPECIFIC VALUES

We can also create arrays with any specific values. Let's say we want to create a large array of 100 by 100 elements consists of 0s, and another one with 1s. The functions we could use here are "zeros" and "ones," respectively (Figs. 1.10 and 1.11).

If a NumPy array has more than 1000 elements, it drops the middle rows and columns from the output display, as shown in Fig. 1.10. For checking the shape of an array, the "`.shape`" function can be used (Fig. 1.11).

We may also use "`ones`" or "`zeros`" and the "`reshape`" function together in one line. Figure 1.12 shows how to use "`ones`" and "`reshape`" functions in one line to create a 100 by 100 array.

```
In [16]: C.shape
Out[16]: (100, 100)
```

Figure 1.11: Using "`.shape`" function to check an array shape.

```
In [17]: D = np.ones(10000).reshape(100,100)

In [18]: D
Out[18]: array([[1., 1., 1., ..., 1., 1., 1.],
                [1., 1., 1., ..., 1., 1., 1.],
                [1., 1., 1., ..., 1., 1., 1.],
                ...,
                [1., 1., 1., ..., 1., 1., 1.],
                [1., 1., 1., ..., 1., 1., 1.],
                [1., 1., 1., ..., 1., 1., 1.]])
```

Figure 1.12: Using "`ones`" and "`reshape`" functions in one line to create a 100 by 100 array.

```
In [19]: E = np.full((3,10),55)

In [20]: E
Out[20]: array([[55, 55, 55, 55, 55, 55, 55, 55, 55, 55],
                [55, 55, 55, 55, 55, 55, 55, 55, 55, 55],
                [55, 55, 55, 55, 55, 55, 55, 55, 55, 55]])
```

Figure 1.13: Using "`full`" function to create a 3 by 10 array where all elements are 55.

For creating an array filled with a different number (e.g., 55), we may use the "`full`" function. The following commands in Fig. 1.13 are creating a 3 by 10 array where all elements are 55.

1.4.4 BROADCASTING IN NUMPY

If we have arrays with different shapes that don't allow mathematical operations, then changing the arrays based on the sizes is known as broadcasting. This operation is convenient, but sometime it may give misleading output. Therefore we have to track the array shapes frequently while doing matrix operations in our programming.

NumPy always tries to perform the broadcasting if the shapes of arrays are different. If we use a scalar or single value, it will be broadcasted as a same shaped array. For example, if we add 5 to `E` (3 by 10 array in Fig. 1.13), 5 will be broadcasted, and will be added to all the elements of `E`. This broadcasting operation is shown in Fig. 1.14.

```
In [21]: E+5

Out[21]: array([[60, 60, 60, 60, 60, 60, 60, 60, 60, 60],
                [60, 60, 60, 60, 60, 60, 60, 60, 60, 60],
                [60, 60, 60, 60, 60, 60, 60, 60, 60, 60]])
```

Figure 1.14: Scalar value 5 is added to E (3 by 10 array). Here, 5 is broadcasted and added to all the elements of E.

```
In [22]: F = np.full((3,1),2)

In [23]: F

Out[23]: array([[2],
                [2],
                [2]])

In [24]: E*F

Out[24]: array([[110, 110, 110, 110, 110, 110, 110, 110, 110, 110],
                [110, 110, 110, 110, 110, 110, 110, 110, 110, 110],
                [110, 110, 110, 110, 110, 110, 110, 110, 110, 110]])
```

Figure 1.15: NumPy broadcasting operation during multiplication of two different array sizes.

```
In [25]: np.dot(E,F)

---------------------------------------------------------------------------
--
ValueError                                Traceback (most recent call las
t)
<ipython-input-25-cbafb301adce> in <module>
----> 1 np.dot(E,F)

<__array_function__ internals> in dot(*args, **kwargs)

ValueError: shapes (3,10) and (3,1) not aligned: 10 (dim 1) != 3 (dim 0)
```

Figure 1.16: The NumPy dot multiplication will give an error since the array sizes of E and F are different.

Now let's say we have another array of "2" (Fig. 1.15) whose size is 3 by 1. If we multiply E and F, only the column of F will be broadcasted horizontally to get the multiplication done without any error.

Here, if we want to do the dot multiplication (using `np.dot` function), we will get an error because the sizes of these two matrices are not aligned (Fig. 1.16).

To fix this error, we have to make sure that the sizes of the matrices are such that actual matrix multiplication is allowed. Therefore, the column size of the first matrix should be equal to

```
In [27]: F = np.full((10,3),2)

In [28]: F

Out[28]: array([[2, 2, 2],
                [2, 2, 2],
                [2, 2, 2],
                [2, 2, 2],
                [2, 2, 2],
                [2, 2, 2],
                [2, 2, 2],
                [2, 2, 2],
                [2, 2, 2],
                [2, 2, 2]])

In [29]: np.dot(E,F)

Out[29]: array([[1100, 1100, 1100],
                [1100, 1100, 1100],
                [1100, 1100, 1100]])
```

Figure 1.17: Reshape `F` to do the dot multiplication of `E` and `F`.

the row size of the second matrix. Now, let's reshape the matrix size `F` using the "`full`" function, as shown in Fig. 1.17 and then try to do the `dot` multiplication.

CHAPTER 2

Introduction to PyTorch

2.1 INTRODUCTION

PyTorch is a highly optimized library that can manipulate data in the form of tensors. This library was introduced in 2016, and since then, it has been used in creating deep learning models effectively using GPUs and CPUs. The PyTorch documentation and libraries can be found at [4].

For multi-dimensional data manipulation, machine learning architectures use tensors as their primary data structure. François Chollet [5] defined tensor as a container of data, and also classified typically used multi-dimensional tensors in neural-networks. A Tensor can be 0-D (only one scalar value), 1-D (one-dimensional vector), 2-D (two-dimensional matrix), 3-D (three-dimensional array to represent three channels of RGB for color images), 4-D (four-dimensional array, which can be a collection of RGB color images), and 5-D collection of RGB images represented by four-dimensional tensors at different times. Although the concept of a tensor is different in the field of physics and engineering, in recent years, the machine learning community widely adopted this term to explain generalized matrices. So, in this book, we will consider tensor as a vehicle to organize information in more than two dimensions. The multi-dimensional tensors can be illustrated by using cubes, as shown in Fig. 2.1.

In deep learning architecture, the tensor-based data can quickly become enormous during weight and back-propagation calculations. Hence, most of the models are executed on GPUs for parallel computing and effective data manipulations. In 2018, Google released TPU (Tensor Processing Units), which is specially designed for Google's Tensorflow software to train deep neural networks [6].

2.2 SETTING UP PYTORCH

PyTorch is relatively easy to learn, especially by someone who already knows a few NumPy operations. Both NumPy and PyTorch use similar syntaxes. The "`numpy.zeros( )`" produces an array of zeros, "`numpy.ones( )`" produces an array of ones, and `numpy.random.rand( )` produces an array of random numbers. Similarly, when we use PyTorch, we can use "`torch.zeros( )`" to produce a tensor of zeros, "`torch.ones( )`" to produce a tensor of ones, and "`torch.rand( )`" to produce a tensor of random numbers.

Before running PyTorch in Jupyter Notebook, it is important to change the programming environment using Anaconda Prompt (Anaconda3). The procedure of changing the environment in Anaconda prompt, and running Jupyter Notebook are given in the following steps.

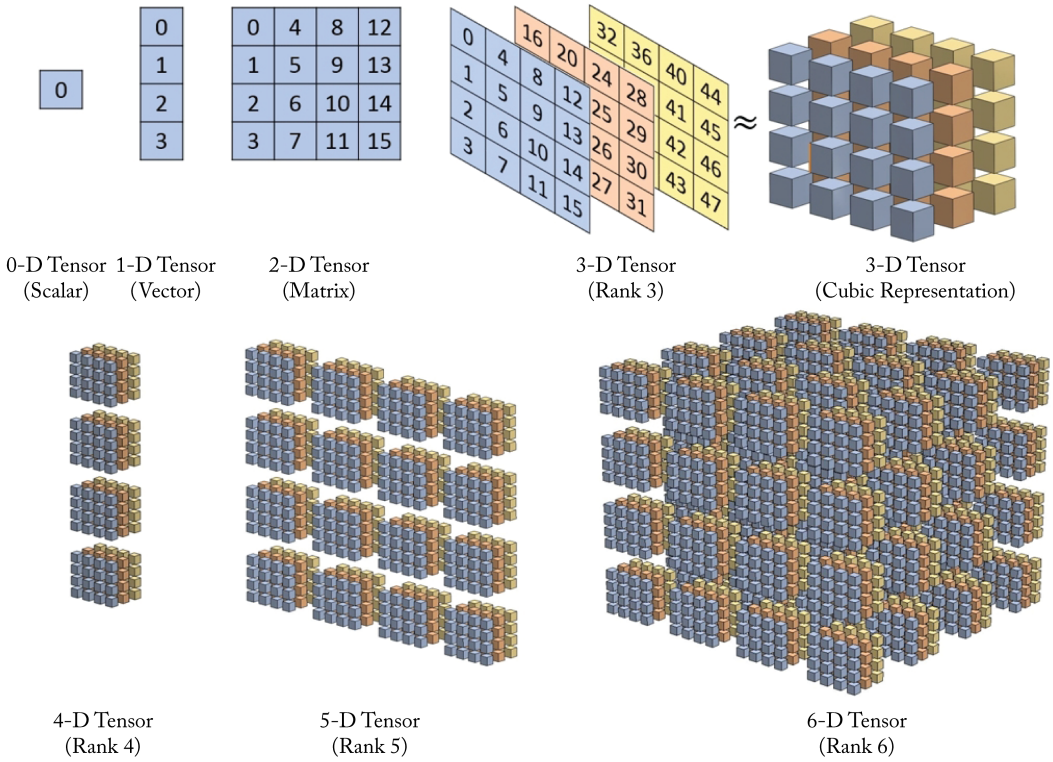

Figure 2.1: Illustration of multi-dimensional tensors (0-D to 6-D).

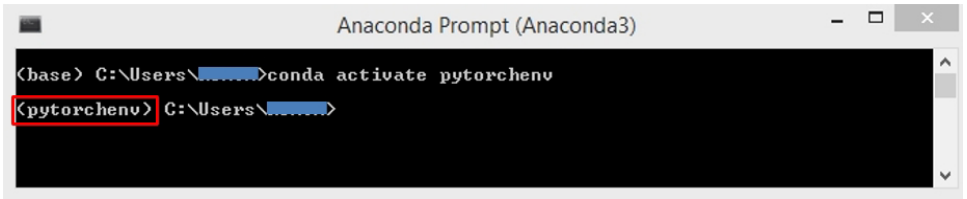

Figure 2.2: Changing Anaconda's environment for PyTorch. The blue lines here are for hiding usernames.

Step 1: Go to the search bar, type Anaconda prompt, and select "Anaconda Prompt."

Step 2: Note that the "`<base>`" environment is written before the path.

Step 3: Type "conda activate pytrochenv" and press Enter (Fig. 2.2). Note that the "`<base>`" has been changed to "`<pytorchenv>`."

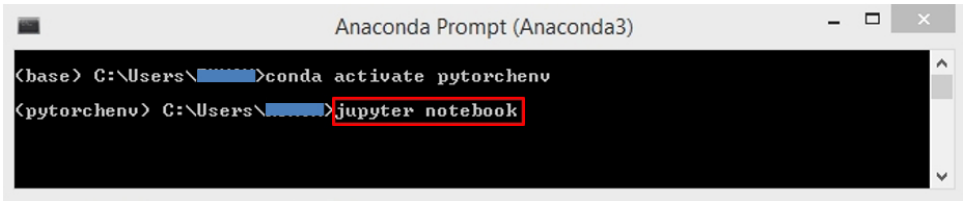

Figure 2.3: Type "jupyter notebook" in Anaconda Prompt and press Enter.

Figure 2.4: Jupyter Notebook interface. Here, select new to create a new script in PyTorch environment.

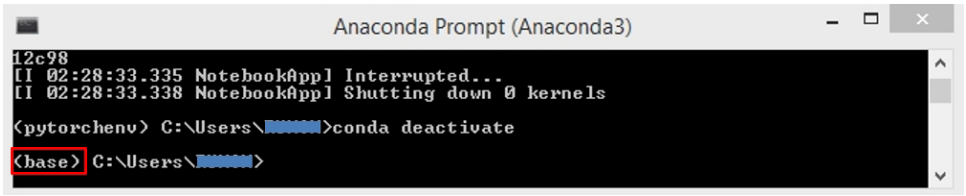

Figure 2.5: Type "conda deactivate," and press Enter to deactivate PyTorch and go back to the base environment.

Step 4: To open Jupyter Notebook type "jupyter notebook," and press Enter (Fig. 2.3).

Step 5: Now, the Jupyter Notebook will start, and we will be able to create a new python file for using the PyTorch functionalities (Fig. 2.4).

Step 6: Once we finish using Jupyter Notebook, we can shut down the kernel by pressing "Ctrl+C" in the command window (multiple times might be required), and then we can deactivate the PyTorch environment by typing "conda deactivate" in the command window (Fig. 2.5). Note that in the terminal, the Anaconda environment changes from "`<pytorchenv>`" to "`<base>`."

```
In [1]: import torch
        import numpy as np
```

Figure 2.6: Importing torch and numpy library from Jupyter Notebook.

```
In [2]: A = np.arange(0,18,2)

In [3]: A
Out[3]: array([ 0,  2,  4,  6,  8, 10, 12, 14, 16])
```

Figure 2.7: Creating a NumPy array using "arange" function.

```
In [4]: A = A.reshape(3,3)
        A
Out[4]: array([[ 0,  2,  4],
               [ 6,  8, 10],
               [12, 14, 16]])

In [5]: A.dtype
Out[5]: dtype('int32')
```

Figure 2.8: Reshape `A` and check the data type.

2.3 BASIC PYTORCH OPERATIONS

Now, let's create some tensors in Jupyter Notebook using PyTorch, and compare their data types with NumPy. First of all, we need to import "`torch`" and "`numpy`" library, as shown in Fig. 2.6.

Note that if you hit enter in Jupyter Notebook, it will go to the next line without execution. So, after typing two or multiple lines (as needed), you may press "Shift+Enter" to execute those lines only.

Now, let's create an array (Fig. 2.7) and then convert it to a 2D shape.

Here the "`numpy.arange( )`" has a start value (0) and stop value (18). And the "2" at the end is the interval between increments. So NumPy will create an array from 0 to 18 (excluding). To reshape it to a 3 by 3 (2D array), use the "`reshape`" function (Fig. 2.8). If you check the data type of `A` by using "`dype`" command, you will see that it is an "`int32`" NumPy array.

Now, let's convert `A` to a PyTorch tensor using the torch commands shown in Fig. 2.9. After using "`torch.from_numpy`" you will see that the data type remains integer `(torch.int32)`. To convert it to a PyTorch tensor function, you will need to assign it to a variable name (e.g., input [7] in Fig. 2.9).

Note that you may use "`torch.arange`" to create a tensor directly (Fig. 2.10).

```
In [6]: torch.from_numpy(A)

Out[6]: tensor([[ 0,  2,  4],
                [ 6,  8, 10],
                [12, 14, 16]], dtype=torch.int32)

In [7]: X = torch.from_numpy(A)
        X.type

Out[7]: <function Tensor.type>
```

Figure 2.9: Creating tensors from a NumPy array.

```
In [8]: B = torch.arange(0,18,2)
        B
        B.type

Out[8]: <function Tensor.type>

In [9]: if torch.cuda.is_available():
            my_device = torch.device("cuda:0")
```

Figure 2.10: Use "`torch.arange`" to create a tensor and use "`torch.device`" to define GPU/CUDA.

```
In [10]: torch.cuda.is_available()

Out[10]: False

In [11]: torch.cuda.device_count()

Out[11]: 0

In [12]: B.device

Out[12]: device(type='cpu')
```

Figure 2.11: Check GPU/CUDA availability and device type in PyTorch.

Using a GPU for running a deep learning model is highly recommended, and if you want to define a GPU/CUDA, you can use "`torch.device( )`" function, as shown in Fig. 2.10. Right now, the system we are using doesn't have a GPU. We can quickly check the status of GPU by using the commands shown in Fig. 2.11. We can also check the device type for tensor `B`, which is a CPU in our example.

CHAPTER 3

Basic Artificial Neural Network and Architectures

3.1 INTRODUCTION

An artificial neural network is an algorithm or software architecture that has interconnected layers of functions similar to the biological neurons. Although the process of learning in the human brain is super complicated and not fully understood, the software architecture of machine learning uses activation function and connected pathways that can be compared to the biological neurons and synapses, respectively. Furthermore, an artificial neural network can learn (alternatively, find optimum model weights or biases) through training, which is an important phenomenon that mimics the process of learning in our human brain. For these reasons, the term "Neural" is used in the field of machine learning.

The neural network architecture consists of several basic components, such as the number of inputs, outputs, hidden layers, and the type of activation functions. A simple multi-layer neural network has a topology similar to Fig. 3.1.

The hidden layer inside a network contains neurons that perform tensor operations based on the output requirements. For example, if our required output is a numerical value, such as predicting body temperature or test score, we use linear regression. On the other hand, we can use logistic regression to predict binary outputs and multi-class outputs. Binary outputs are predicting between true or false, pass or fail, cat or non-cat picture, and multi-class outputs are predicting one from multiple classes, such as finding out if a picture is a cat, bird, dog, or fish picture. The logistic regression can also be used for predicting ordinal outputs (e.g., determining high, medium, or low ranges). Each input in the neurons of the input layer is multiplied by weights associated with its connecting pathways and then added to a bias to generate the next output.

The learning process of an artificial neural network is actually finding out weights and biases that will generate a minimum error. The weights and biases always needed to be adjusted if the network produces poor results. When the network finds the best or desired result, it will not update these parameters.

Several methods of learning schemes are widely used in machine learning based on data types or observation methods such as supervised, unsupervised, and reinforcement learning. When the network can compare its predictions with the correct results to make adjustments, then it is called supervised learning. Unsupervised learning is the type of learning when the

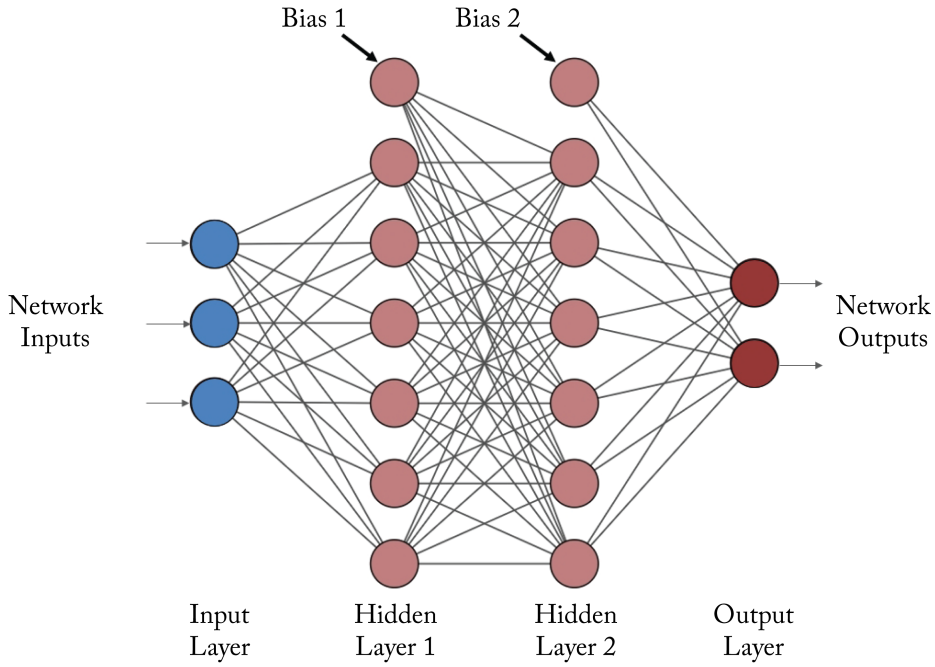

Figure 3.1: Multi-layer neural network architecture with two hidden layers.

correct results of the example dataset are unknown. And, the reinforcement learning are used when learning through observations from a complex environment is required. Here, the model makes a final decision based on the penalties or rewards it received for its performance.

The selection of these schemes depends on the types of dataset and output requirements for a particular task.

3.2 APPLICATIONS

The artificial neural network is very effective in function approximations, identifying patterns, and implementing complex mapping. Recently, it has been used in diverse science, engineering, art, and medical applications. It is also widely used for forecasting and predicting many real-life businesses, economic and social issues [7–10]. In Chapter 7, we provided a case study of a multiclass classification problem where the model learns from 8,144 car images to determine car model, make, and year.

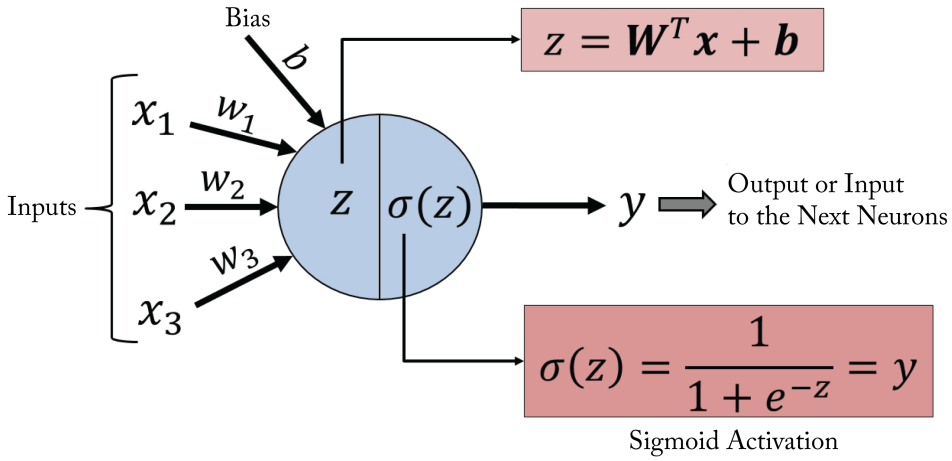

Figure 3.2: Computational steps inside one neuron of the hidden layer. Here, $\sigma(z)$ is the sigmoid activation, x_1, x_2, x_3 are inputs to the neuron, w_1, w_2, w_3 are the weights, and b is the bias to the input.

3.3 NEURONS AND ACTIVATION FUNCTIONS

Many internal parameters can influence the output found from an artificial neural network. But generally, outputs are directly related to the number of neurons and their activation functions.

The neurons are the nodes, typically shown by circles that are arranged inside the layers. Inside a layer, these neurons take input weights from all connections and add a bias, initializes these parameters, and then calculates the output using another function known as an activation function. The mathematical operations happening inside each layer amplify dramatically as the neurons and connections are increased. Figure 3.1 shows a simple neural network when we have two hidden layers, and each of them containing four neurons. Figure 3.2 focuses on the operations happening inside one of the neurons of hidden layer 1.

This operation can be divided into two steps of computations. First, $\boldsymbol{Z}$ is calculated using the weights ($\mathbf{W}$), bias ($\mathbf{b}$), and inputs ($\mathbf{x}$). Second, the activation function $\boldsymbol{\sigma}$ is calculated using $\boldsymbol{Z}$. We could simply say that output of a neuron is a function of a function. In Fig 3.2, a sigmoid activation function is shown as an example.

The sigmoid activation function (shown in Fig. 3.2) has the ability to map any predicted value between 0 and 1. This output can be used to estimate the probability, where a threshold or tipping point (e.g., 0.5) is used for binary classification problems. For example, if $output \geq 0.5$, class $= 1$ and if $output < 0.5$, class $= 0$. There are many other types of activation functions available for an artificial neural network. It is one of the important selection parameters that dictate the computational efficiency.

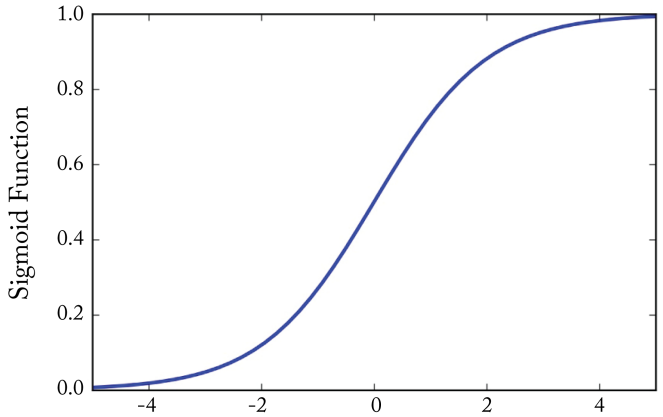

Figure 3.3: Plot of Sigmoid activation function in between −5 to +5.

The use of sigmoid activation in a deep neural network is minimal, and most of the time, nonlinear activations such as tanh, ReLU, and Softmax are used. It is important to note that different types of activations can exist in the neurons of the hidden and output layer. Only the input layer neurons don't have activation functions. The derivatives of activation functions also play an important role as they are used in the backpropagation error calculation during gradient descent. A short description of typical activation functions and their derivatives are given in the following sections for reference.

3.3.1 SIGMOID ACTIVATION

The sigmoid activation and its derivative are defined by Equations (3.1) and (3.2), respectively:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{3.1}$$

and

$$\sigma'(z) = \sigma(z)(1 - \sigma(z)). \tag{3.2}$$

This function is also known as the logistic regression function. We can use it when we need to determine the probability between 0 and 1. Sometimes, the sigmoid function exists only in the output layer of a neural network for doing the binary classification. A plot of sigmoid function in the range of −5 to 5 is given in Fig. 3.3.

3.3.2 HYPERBOLIC TANGENT ACTIVATION

Hyperbolic tangent, also known as tanh function and its derivative are defined by Equations (3.3) and (3.4), respectively:

$$\sigma(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{3.3}$$

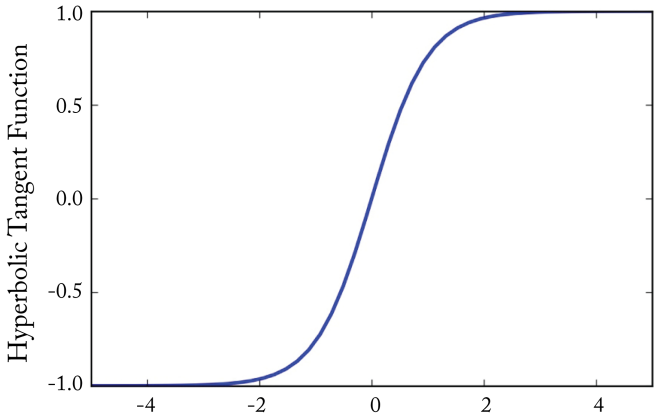

Figure 3.4: Plot of hyperbolic tangent activation function in between −5 to +5.

and

$$\sigma'(z) = \left(1 - \sigma(z)^2\right). \tag{3.4}$$

This function performs better than the sigmoid function when the output is a large negative number. This function can also map numbers between −1 to +1. A plot of hyperbolic function in the range of −5 to 5 is given in Fig. 3.4.

3.3.3 RECTIFIED LINEAR UNITS (RELU) ACTIVATION

The Rectified Linear Units, also known as ReLU activation, and its derivatives are defined by Equations (3.5) and (3.6), respectively:

$$\sigma(z) = \begin{cases} 0 & \text{for } z \leq 0 \\ z & \text{for } z > 0 \end{cases} \tag{3.5}$$

and

$$\sigma'(z) = \begin{cases} 0 & \text{for } z \leq 0 \\ 1 & \text{for } z > 0. \end{cases} \tag{3.6}$$

The ReLu function updates parameters faster as its gradient calculation is simpler compared to other activations. A plot of ReLU function in the range of −1 and +1 is given in Fig. 3.5.

3.3.4 LEAKY RECTIFIED LINEAR UNITS (LEAKY RELU) ACTIVATION

The Leaky Rectified Linear Units, also known as Leaky ReLU, is a type of improved ReLU activation that doesn't have 0 slopes for negative values. Leaky ReLU and its derivatives can be

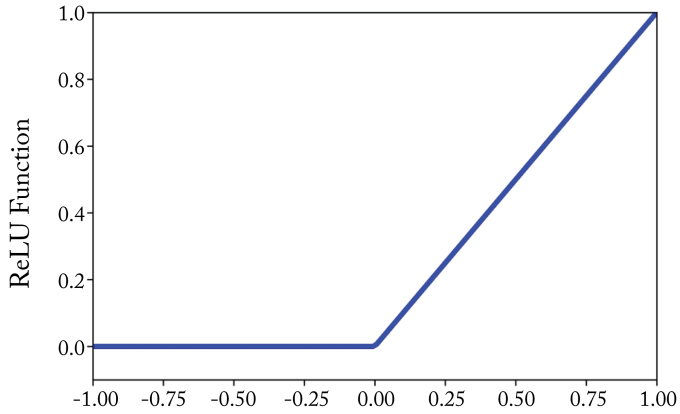

Figure 3.5: Plot of ReLU activation function in between -1 to $+1$.

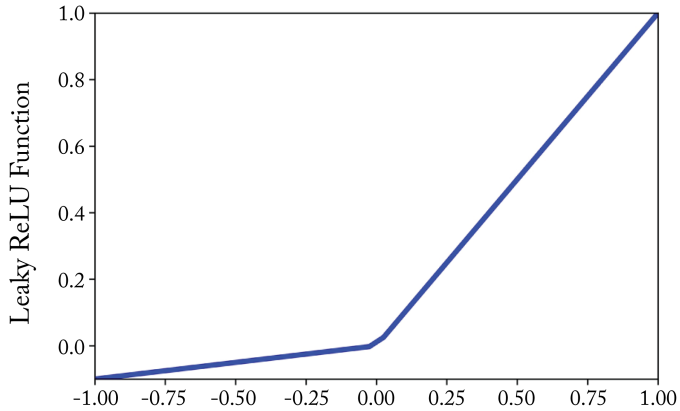

Figure 3.6: Plot of Leaky ReLU activation function in between -1 to $+1$.

defined by Equations (3.7) and (3.8), respectively:

$$\sigma(z) = \begin{cases} az & \text{for } z \leq 0 \\ z & \text{for } z > 0 \end{cases} \tag{3.7}$$

and

$$\sigma'(z) = \begin{cases} a & \text{for } z \leq 0 \\ 1 & \text{for } z > 0. \end{cases} \tag{3.8}$$

A plot of Leaky ReLU function in the range of -1 and $+1$ is given in Fig. 3.6.

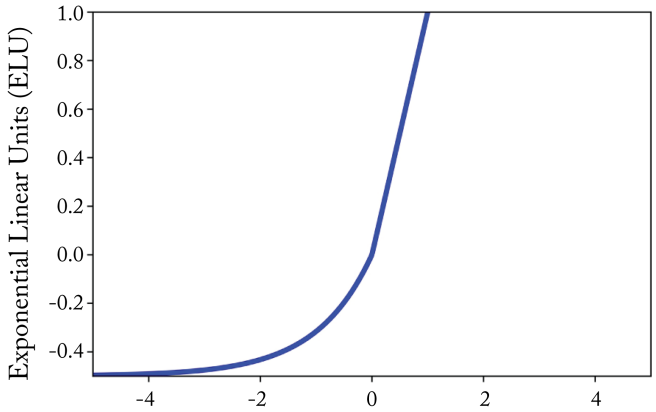

Figure 3.7: Plot of ELU activation function in between −5 to +5.

3.3.5 EXPONENTIAL LINEAR UNITS (ELU) ACTIVATION

The Exponential Linear Units (ELU) and its derivatives can be defined by Equations (3.9) and (3.10), respectively:

$$\sigma(z) = \begin{cases} a\,(e^z - 1) & \text{for } z \leq 0 \\ z & \text{for } z > 0 \end{cases} \tag{3.9}$$

and

$$\sigma'(z) = \begin{cases} f(z) + a & \text{for } z \leq 0 \\ 1 & \text{for } z > 0. \end{cases} \tag{3.10}$$

ELU activation function converges faster in learning compared to other functions, and it pushes the mean of the activations closer to zero. A plot of ELU function in the range of −5 to +5 is given in Fig. 3.7.

3.3.6 SOFTPLUS ACTIVATION

The Soft Plus activation function and its derivative are defined by Equations (3.11) and (3.12), respectively:

$$\sigma(z) = \ln(1 + e^x) \tag{3.11}$$

and

$$\sigma'(z) = \frac{1}{1 + e^{-x}}. \tag{3.12}$$

The Softplus function is smoother than the ReLU function, and it's derivative becomes a sigmoid function. But due to exponential operation, it is not as fast as ReLU activation. A plot of Softplus activation in the range of −5 to +5 is given in Fig. 3.8.

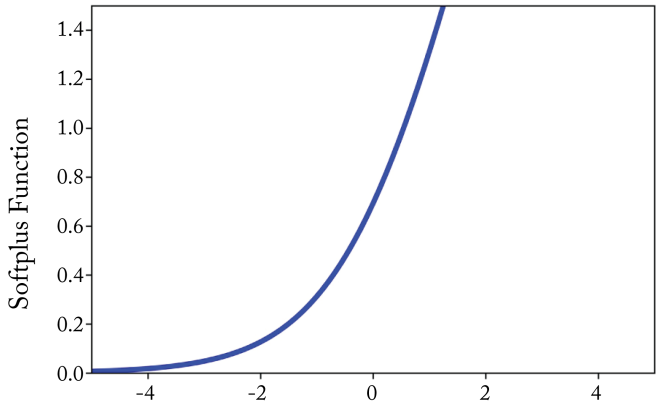

Figure 3.8: Plot of Softplus activation function in between −5 to +5.

Finding out the appropriate activation function for a model is a challenge due to very small gradients during the backpropagation steps. This phenomenon is also known as diminishing gradients. Besides the activation functions shown here, many other functions are continuously being explored and added in the field of machine learning. For example, ArcTan, Piecewise Linear Unit (PLU), Scaled Exponential Linear Unit (SELU), Inverse Square Root Unit (ISRU), etc. A comprehensive list of activation functions can be found on the Wikipedia page [11].

3.4 MINIMIZING THE LOSS FUNCTION

The loss function is the quantity that will be minimized during training [5]. It is also known as the objective function or error function. A neural network may have multiple loss functions based on the number of outputs. A classification model should have a cross-entropy loss or a log loss of 0. Generally, for a two-class classification problem binary cross-entropy, and for multiclass classification problems, categorical cross-entropy is used. The learning process of a neural network is basically finding out weights and biases to minimize the loss function.

Typically, the loss function is also called cost function if the error is calculated for the entire training set or a batch. The standard binary cross-entropy loss function can be defined by Equation (3.13) [12]:

$$J = -\frac{1}{M}\sum_{m=1}^{M}\left[y_m \times \log\left(h_\theta\left(x_m\right)\right) + \left(1 - y_m\right) \times \log\left(1 - h_\theta\left(x_m\right)\right)\right], \tag{3.13}$$

where M = number of training examples, y_m = target label for training example m, x_m = input for training example m, h_θ = model with neural network weights θ. For categorical targets if $\hat{y}_1 \ldots \hat{y}_m$ are the probabilities of the "m" classes, and the rth class is the ground-truth class, then

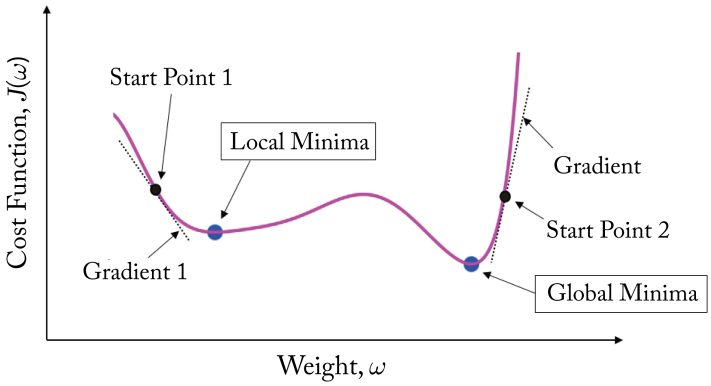

Figure 3.9: If the initial starting point is Start Point 1, the gradient descent algorithm will converge to a local minima. On the other hand, if the Start Point 2 is used, the algorithm will converge to the global minima.

the loss function for a single instance is defined by Equation (3.14) [13]:

$$J = -\log(\hat{y}_r). \tag{3.14}$$

The goal of any neural network is to minimize Equation (3.14). This minimization process is done by updating weights using an optimization function or optimizer. An important iterative algorithm known as backpropagation is used to apply this optimization, and in this process, the loss function dictates how well the network is learning.

Many different optimizers have been used in artificial neural networks such as gradient descent, gradient descent with momentum, Root Mean Square Propagation (RMSProp), Adam optimizers, and so on. The selection of a suitable optimizer depends on the computational scenario, and they typically differ in terms of their simulation speed and convergence ability.

3.5 GRADIENT DESCENT ALGORITHM

The gradient descent is an iterative first-order optimization algorithm that helps the neural network to minimize the cost function, J. In the case of linear regression, we can use derivatives to find a minimum of convex functions. But in the artificial neural network, we always deal with a non-convex function, and therefore gradient descent algorithm can end up finding the local minima instead of global minima.

Figure 3.9 shows how the gradient descent can end up in different minima based on the starting point. This example is for only one weight parameter, ω. For a 3D surface plot where the cost function J is a function of two parameters (ω and b) and forms a non-convex shape, the gradient descent algorithm may converge to a local minima instead of a global one as shown

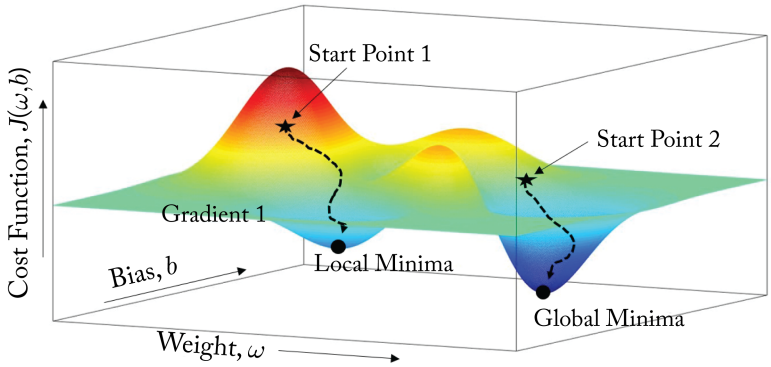

Figure 3.10: If the Cost function, J is a function of two parameters (weight and bias), the gradient descent algorithm will search for a minima on the surface. In this example plot, the Start Point 1 will converge to a local minima, and the Start Point 2 will converge to a global minima.

in Fig. 3.10. Here, the height of the surface shows the value of cost function J for respective weight and bias.

In most of the machine learning optimization tasks, the cost function is a higher-dimensional function due to the existence of many weights and biases. This type of higher-dimensional plot is impossible to visualize.

The gradient descent algorithm is simultaneously performed for all values of weights and biases. If we have n neurons in a hidden layer, all the weights will update through Equation (3.15) [14]. Here α is the learning rate, which tells the algorithm how big or small the step size would be while moving in the direction of downhill steep. The negative gradients push the algorithm to move toward the downhill directions.

$$\begin{aligned}
\omega_1 &= \omega_1 - \alpha \frac{\partial}{\partial \omega_1} J(\omega, b) \\
\omega_2 &= \omega_2 - \alpha \frac{\partial}{\partial \omega_2} J(\omega, b) \\
\omega_3 &= \omega_3 - \alpha \frac{\partial}{\partial \omega_3} J(\omega, b) \\
&\vdots
\end{aligned} \tag{3.15}$$

.
.

$$\omega_n = \omega_n - \alpha \frac{\partial}{\partial \omega_n} J(\omega, b).$$

The bias parameters for a hidden layer will update using Equation (3.16):

$$b = b - \alpha \frac{\partial}{\partial b} J(\omega, b). \tag{3.16}$$

Equations (3.15) and (3.16) shows the calculation for one layer only. This computational process becomes expensive in case of deep learning when we have many hidden layers.

The learning rate, α is one of the critical hyper-parameters for model training, and it needs to be tuned carefully. If the learning rate is too low, the gradient descent algorithm will become computationally expensive. On the other hand, if the learning rate is relatively big, it may overshoot the minima and fail to converge.

One way to use the learning rate effectively is to reduce its value as the training progresses. This process is known as the adaptive learning rate [15]. The adaptive learning rate can be reduced or adjusted using a pre-defined schedule such as time-based decay, step decay, and exponential decay [16]. Many algorithms, such as Adam, SGD, AdaGrad, and RMSProp, use the adaptive learning rate. The PyTorch "`torch.optim`" package has built-in functions that allow users to pass arguments in different optimizers for implementing the adaptive learning algorithm [17].

CHAPTER 4

Introduction to Deep Learning

4.1 INTRODUCTION

Deep learning is a computational neural network architecture containing multiple hidden layers. Since deep learning is a larger neural network, it is treated as a subfield of machine learning and artificial intelligence. It is an extremely powerful tool for speech recognition, remote sensing, classifying objects, and pattern recognition tasks. For the past few years, deep learning algorithms are extensively employed in many different scientific and engineering applications such as self-driving cars, fraud detections, health care, entertainment, and so on. Using many hidden layers, a deep learning model learns from big datasets with multiple levels of abstraction.

During the learning process, a backpropagation algorithm helps the model to fine-tune the parameters of one layer from the calculated parameters of the previous layer. Figure 4.1 shows a schematic of a deep learning model with n hidden layers and biases, where every neuron in a layer is connected to every neuron of the next layer.

If we start modeling from scratch, the optimum numbers of layers and neurons of a deep network should be chosen carefully. There are no correct layer or neuron numbers for a specific problem, and numerous factors can influence the output. Many powerful deep learning architectures such as inception-V3, ResNet, Inception-ResNet, DenseNet, and Pyramidal Residual Net have more than 100 layers [18–22]. However, the performance of these architectures depends on both node and layer numbers. Having too many hidden layers is not always effective, and it should be selected based on the type of training dataset and output requirements. An optimum selection of these parameters (also called capacity) can help us to avoid overfitting and underfitting during the training process [23]. Most of the time, we don't need too many layers as they can add unnecessary difficulties to the computational process. A system with 5–20 nonlinear hidden layers can map extremely complex functions for inputs that are simultaneously sensitive to minute details [24]. Therefore, while selecting a deep learning model, we need to be extra cautious regarding network layers and architecture.

As the growth of deep learning is accelerating, many new deep learning models are continuously coming out each year. Some of the popular deep learning architectures are briefly discussed in the following sections.

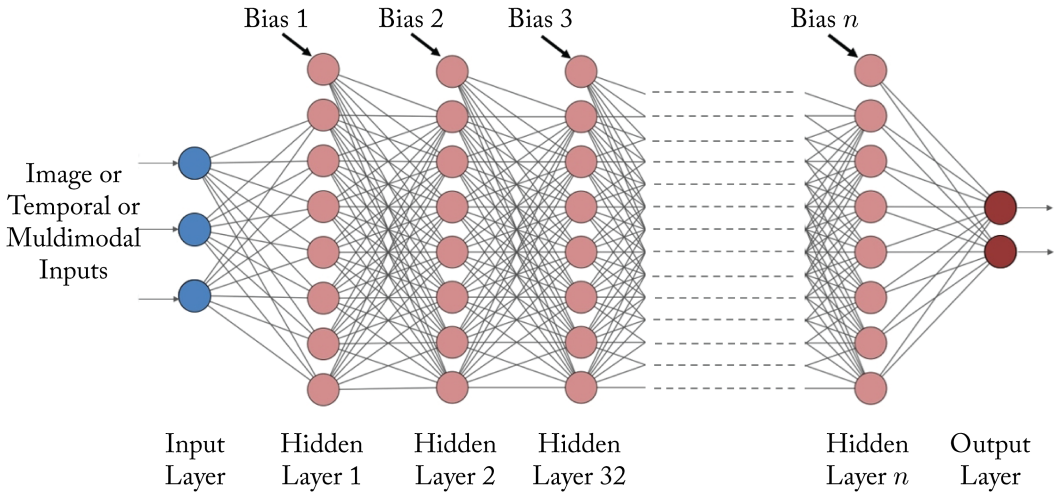

Figure 4.1: Schematic of deep learning model with n hidden layers and biases, where every neuron in a layer is connected to every neuron of the next layer.

4.2 CONVOLUTIONAL NEURAL NETWORK

Convolutional Neural Network (CNN), also known as ConvNet, was initially developed by Yann LeCun for computer vision and image processing [25]. Later on, it has been successfully modified by other researchers for improved sentence classification, speech recognition, image segmentation, and image classification tasks [26–29]. For computer-vision related operations, CNN is the most widely used deep learning method. A CNN will have a number of convolutional and pooling layers at the beginning, and after that, it goes through the fully connected layers.

The image input in the convolutional layer is typically rearranged in multiple arrays or multi-channeled images. For example, a color image is a combination of three-color channels (red, green, and blue), and each of these channels can be presented in a 2D array for the input. Inside the 2D array, the intensities of these color channels are ranged from 0–255.

The convolutional layer also contains kernels to map features from the input images. They typically slice through image depth, numbers of pixels, or zero-padding around a shape to learn from different sections of three channel inputs. When multiple kernels concatenate with each other, they are called filters. Therefore, filters have one more dimension than the kernels. Inside the network, each of the kernels will add a new layer that corresponds to edges, depths, shapes, or textures of the image.

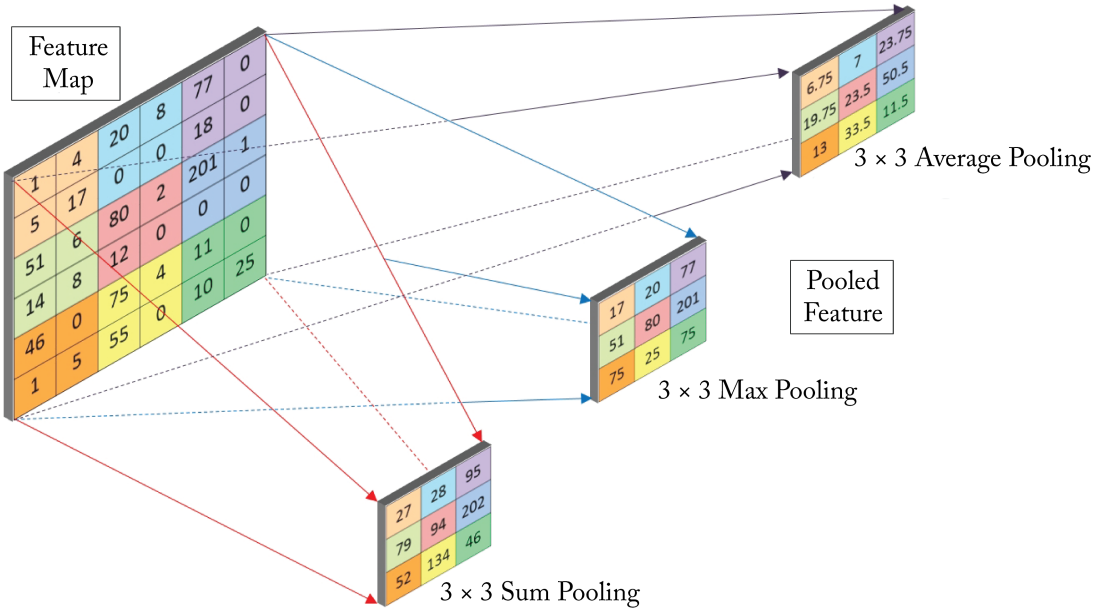

Figure 4.2: Schematic of max, average, and sum pooling methods to remap the input features in a 3 × 3 array.

4.2.1 POOLING LAYER

A pooling layer reduces the spatial size of the representation, which is another integral task of the convolutional neural network. This step is done to pool or extract the most important features from the output of a convolutional layer.

The pooling process can reduce the spatial size or dimension of each feature map through different formulations such as max, average, sum, overlapping, L2, etc. This step generally helps the model to understand feature motifs that may appear at different locations of the training image. A schematic for feature mapping through max-pooling, average-pooling, and sum-pooling is given in Fig. 4.2. Typically, the convolution neural network has several convolutional and pooling layers at the beginning before going through the fully connected layers.

4.2.2 FULLY CONNECTED LAYER

The final stage of the convolutional neural network contains fully connected layers. It is similar to the regular artificial neural network and contains neurons that are connected to all neurons from the previous layer. The training image features that are extracted through convolutional and pooling layers are classified in the fully connected layer. Sometimes, to use computational memory efficiently, two small, fully connected layers are used instead of one large layer. Some of the popular convolution neural networks such as AlexNet, VGG Net uses more than one fully

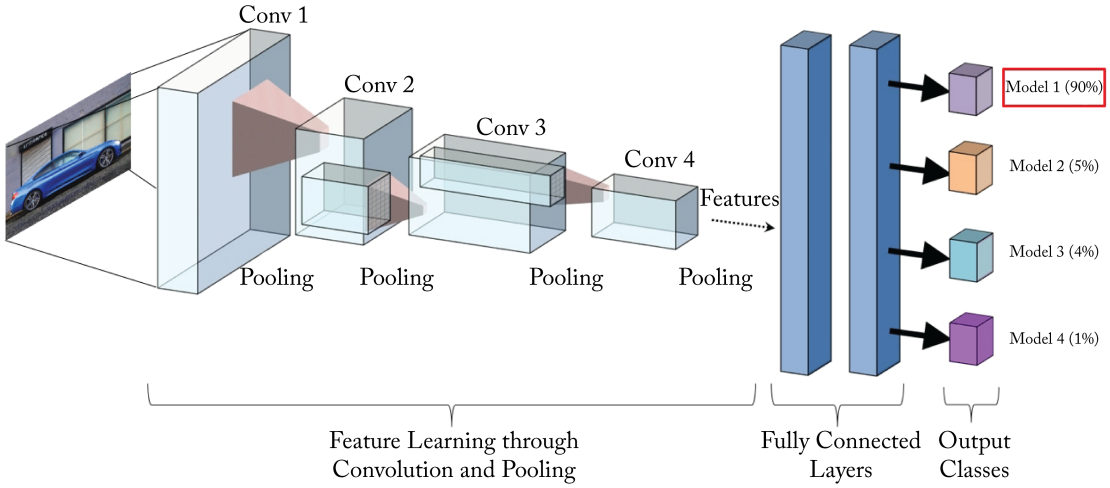

Figure 4.3: CNN transforms the input car image (RGB inputs) through convolutional, pooling, and fully connected layers. The output layer has softmax activation (for multi-class classification) to represent probability distribution for different car models. This example indicates that the car is Model 1.

connected layer, followed by the output layer. The fully connected layers also use a backpropagation algorithm to update their weights and biases. The properties of fully connected layers depend on the types of problems (e.g., classification or segmentation). It is important to note that, for classification and regression problems, another supervised algorithm known as support vector machine (SVM) can also be used instead of fully connected layers.

At the end of the fully connected layer, a softmax activation function can be used for doing multi-class classification, where the probabilities of each class will always sum up to one. Figure 4.3 shows a general framework of a convolutional neural network for multi-class classification problems. This example illustrates that, after training over a lot of images, a deep learning model can learn features to identify a car model based on the given input picture. If we have more than two classes (different car models), the softmax activation function or the SVM can be used in the last layer.

The convolutional model is extensively used in recent years, and it has been proved to be very useful for 2D image classification problems. Very recently, CNN models are used on 3D MRI image data to explore complicated functions of the human brain [30–32]. Among the deep learning architectures, CNN is computationally expensive as it requires a large volume of training data to avoid overfitting.

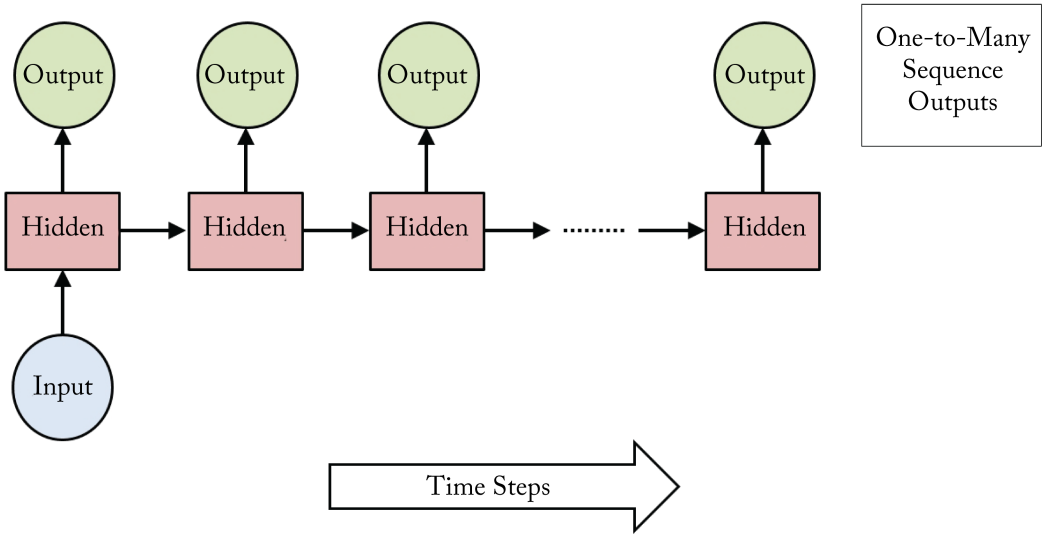

Figure 4.4: Schematic diagram of a one-to-many RNN architecture.

4.3 RECURRENT NEURAL NETWORK (RNN)

The Recurrent Neural Network (RNN) is another popular deep learning architecture that is used for predicting sequential data with the help of an additional memory state. The prediction by RNN architecture depends on both current and past inputs.

The use of RNN escalated rapidly within the last decade. It has been delivering state-of-the-art results in many different fields such as language processing, speech recognition, video analysis, fake video creation, music generation, grammatical inference, trajectory control of robots, and so on [33–39].

4.3.1 BASIC STRUCTURES

RNN deals with vectorize data, i.e., inside this network, all inputs and outputs are vectors for many time-steps. Among many underlying internal architectures of RNN, one-to-many, many-to-one, and many-to-many are commonly used.

For a one-to-many architecture, one input is mapped into multiple outputs (Fig. 4.4). In this case, intermediate representations of the input unit are done in a series of hidden states, where every hidden state is a function of all previous states. An example application of this type of architecture is image captioning, where input is a single image, but the outputs are words of multiple classes.

In the many-to-one RNN architecture, many inputs are mapped into hidden states but produce only one class or quantity output. This type of network is very useful to gauge or classify

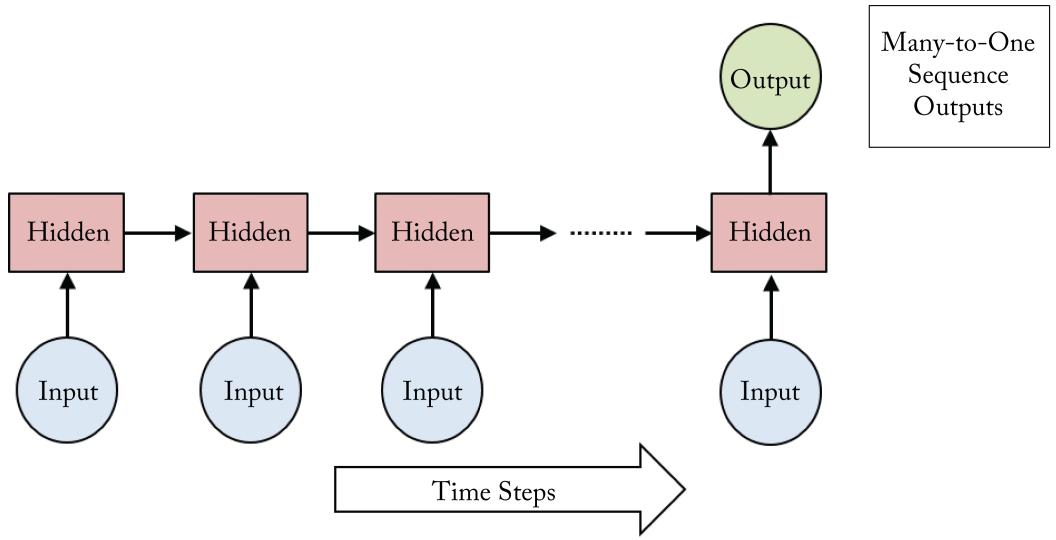

Figure 4.5: Schematic diagram of a many-to-one RNN architecture.

emotional state and sensitivity from writings. A schematic of many-to-one RNN architecture is given in Fig. 4.5.

In the many-to-many RNN structure, multiple inputs go through hidden states and produce multiple outputs. It is used for translating sentences in another language, video frame labeling, and for many other prediction applications where multiple inputs produce multiple outputs. Many-to-many RNN structures might have different layouts based on the numbers of inputs and outputs. A schematic of many-to-many RNN architecture is given in Fig. 4.6.

Besides these RNN architectures, there are many other input-output dynamics. For example, a simple one-to-one RNN structure similar to the classical feed-forward network.

An RNN model is challenging to train because to update the parameters, backpropagation algorithm needs to calculate gradients at different time steps. This operation makes the network unstable due to vanishing and exploding gradients. In order to avoid this problem, different supporting units such as Gated Recurrent Unit (GRU), Bidirectional Recurrent Neural Network (BRNN), and Long-Short-term memory (LSTM) are used. For example, the LSTM network uses cells with input, output, and forget-gate to control the flow of information. LSTM can be used with other tricks such as gradient clipping to set a threshold value for the error gradient and weight regularization (L1-absolute or L2-Squared) to introduce a penalty to the loss functions, etc. An LSTM based recurrent network is more effective when several layers are involved in the sequence of data [34]. To train a recurrent network for complex high-dimensional data, a GPU-based hardware (e.g., NVIDIA cuDNN) that supports deep learning framework is highly recommended.

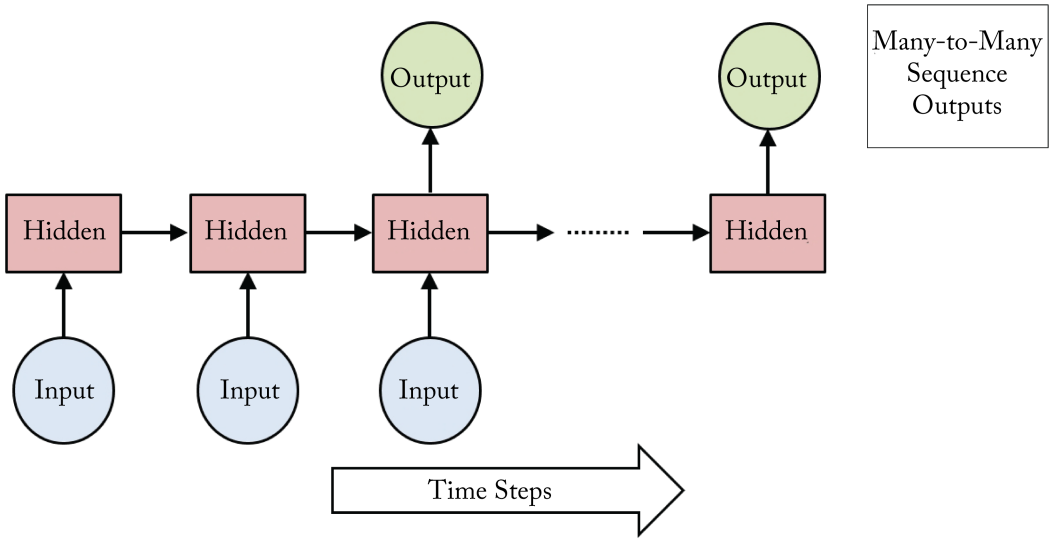

Figure 4.6: Schematic diagram of a many-to-many RNN architecture.

4.4 OTHER DEEP LEARNING MODELS

Among the other deep learning models, recursive and unsupervised pre-trained networks are gaining popularity. Recursive networks apply the same set of weights recursively to learn from structured information, and it is known for being computationally expensive [40]. Recursive models are suitable for pattern analysis as they can process both the numerical and symbolic data together. Different variants of recursive networks are continuously emerging and explored by contemporary researchers. Although it is relatively less popular than the recurrent network, it has shown excellent performance in natural language processing, image decomposition, and forecasting problems.

The unsupervised pre-trained network can help to improve the CNN's performance, and it can reduce labeled data dependency by creating layers of feature detectors [41]. This type of network is suitable for finding hidden patterns inside the data by using autoencoders, Generative Adversarial Networks (GANs), and other methods.

An autoencoder has different variants, but it is generally a neural network where the input features are reduced or mapped in a hidden layer and then restore or remapped to the same numbers of features at the output. The middle layer of an autoencoder, also called the bottleneck is symmetric about the center, and it contains fewer nodes than input or output nodes. A schematic diagram of the autoencoder architecture is given in Fig. 4.7.

The GANs is one of the exciting classes of deep learning. It can produce realistic data by using two parallel adversarial networks that compete with each other. One of the networks

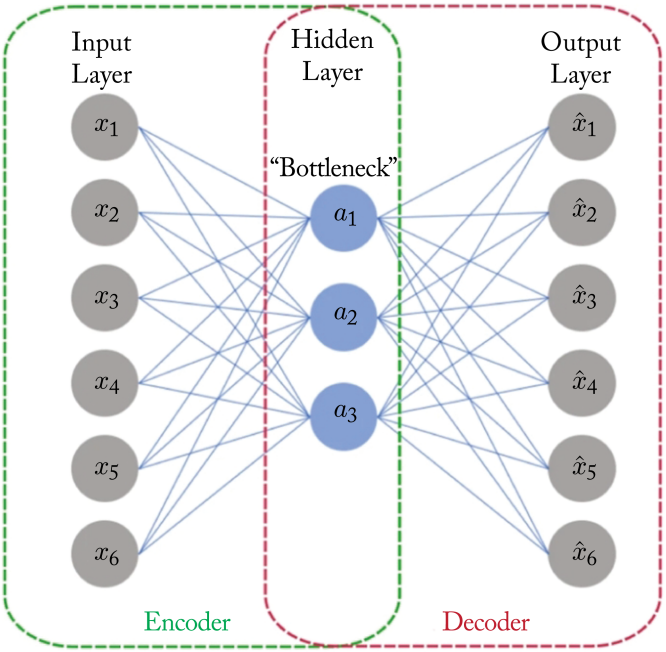

Figure 4.7: Schematic of an autoencoder architecture [42].

works as a data generator, and the other network classifies it by utilizing real data. This process continues until the second network fails to discriminate between the real and synthetic data [13]. A GAN model is relatively difficult to train as it can suffer from mode collapse, diminished gradient, and non-convergence problems [43, 44].

Generative networks can be used to generate realistic pictures or images, and it is explored in many different engineering domains for image reconstructions. Recently, GAN was used to learn scene dynamics from a large amount of unlabeled video (26 terabytes), and it also showed impressive results in recovering features from astrophysical images by reducing random and systematic noises [45].

Although GAN is an amazing development for deep learning, using it, people can make fake media content, realistic images, and online profiles that can negatively affect our society. There are numerous other potential applications of GAN that exist in the field of arts, industry, and medicine, and it will definitely create an impact on our society in the near future.

CHAPTER 5

Deep Transfer Learning

5.1 INTRODUCTION

Transfer learning is a method of applying the knowledge of a pre-trained network to another model when we have insufficient data and computing resources to train a new model from scratch. Using this method, knowledge learned from one type of data can be applied to totally different types of data. The deep convolutional network showed tremendous success in image recognition and classification tasks using transfer learning methods [46, 47]. One of the key challenges to train a deep learning model from the beginning is that it requires a huge amount of labeled dataset, which is not common in many practical scenarios. Generally, a model can make good predictions if it is trained on a big dataset. There are many deep learning models available that are developed from big datasets, such as, ImageNet, MNIST, M.S. COCO, Caltech datasets, Pascal VOC, CIFAR10, and so on. The performance of a deep learning model increases if it is trained on a big dataset. For example, the popular ImageNet dataset contains 14,197,122 images, and the Tina Image contains 79,302,017 images [48, 49]. We can use deep learning models that are trained on these datasets for doing a different task.

5.2 TYPES OF TRANSFER LEARNING

The relationship between traditional machine learning and transfer learning can be categorized into three different ways, according to Pan et al. [50]: Inductive, Transductive, and Unsupervised learning. Inductive transfer learning method can be built by infusing similar instance form different learning tasks, transductive transfer learning can be used when the source and target tasks are the same, and the unsupervised transfer learning do unsupervised learning tasks (e.g., clustering, dimensionality reduction). Pan and Yang [50] also sub-categorized transfer learning approaches based on the instance, feature-representation, parameter, and relational-knowledge transfers.

5.3 USING PRE-TRAINED NETWORKS

If we want to use a dataset for supervised learning, we will need to label or annotate data for training. Annotating a large volume of data, which is an integral part of deep learning, would be extremely time-consuming. Fortunately, as we mentioned in the introduction, there is an easy way to avoid this by implementing knowledge from a pre-trained model architecture that has already been trained on a different dataset. There are many pre-trained models available

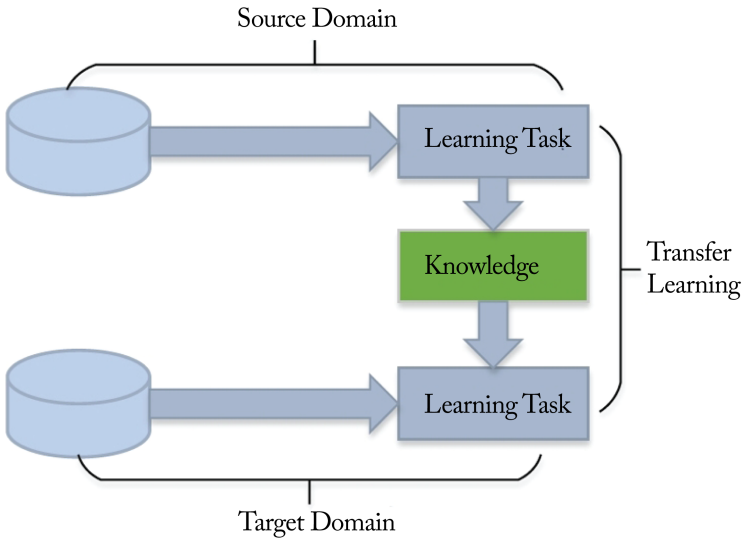

Figure 5.1: Learning process through transfer learning [55].

that we can use effectively, and the amount of effort required for this process is also relatively insignificant. Some of the widely used pre-trained networks are ResNet, GoogleLeNet, VGG, Inception, DenseNet, VGG, EfficientNet, etc. [51]. Many other ready-to-use pre-trained models are available now, and as the field of deep learning is emerging, new models are continuously being tested and added to this list.

One of the advantages of using pre-trained networks is that the architectures have already been trained and tested using a large dataset (e.g., ImageNet). We can just use these networks partially or wholly in the transfer learning process. For partial uses of a model, only the last few layers are regulated to train it on a new dataset, but the core part of the network doesn't change. On the other hand, we can also utilize the whole architecture without parameter values, initialize all the weights randomly, and then train the model using the new dataset. These simple techniques can significantly boost the performance of a deep learning model, and recently, this practice is becoming well accepted in computer vision applications [52–54]. A schematic of transfer learning process is shown in Fig. 5.1.

5.3.1 FEATURE EXTRACTION, FINE TUNING, AND DATA AUGMENTATION

Deep learning architectures learn features through layer by layer. For example, when a convolutional network learns from images, it tends to learn patterns, textures, edges, and brightness in the first few layers. These features from images are used for processing many different types of natural images, and therefore they can be reused. Typically, the last few layers of a deep learn-

ing network are very specific to the current task. So, for a classification problem, if we utilize a pre-train model for transfer learning without using its last layer, we will easily get the hidden states of the trained network to extract some standard features.

For a more rigorous training process, we can reconfigure some of the layers of the model (especially last few) before retraining it on the new data. This step will fine-tune the whole model, but the successful implementation of this approach depends on the dataset and the prediction task we are trying to accomplish. The PyTroch package "torchvision" has many popular model architectures that we can use for deep transfer learning [56].

If we have relatively less amount of data for deep learning, we can use the data augmentation technique to improve model performance. Augmentation will manipulate the training images through different ways, such as flipping, rotating, shearing, cropping, shifting RGB channels, and so on. As a result, we will get various unique and relevant images from a single image. It will increase the amount of training data representation of our model. Recently, different data augmentation techniques are explored by researchers for improving the validation accuracy of a model. We will show a practical implementation of using pre-trained weights, fine-tuning, and data augmentation in Chapter 7.

5.4 MODEL EVALUATION

During training, we will initially need to split our annotated data sets into training and test sets. The training data (can be subdivided into training and validation set) will be used to learn the model parameters, and test data will estimate the accuracy of the learned model. Typically, a large test set will do a good estimation with low variance. There are different types of metrics available that used in model evaluation, such as confusion matrix, accuracy, F1 score, and so on.

To evaluate the model, we can initially check classification accuracy, especially when we have an almost equal number of samples for different classes. It is just the ratio correct predictions to the total numbers of predictions. The confusion matrix can be used to give the quality of output result shown in a matrix, where higher diagonal values indicate many correct results. A confusion matrix can be constructed by finding true positive (model predicted positive, which is a true result), false positive (model predicted positive, which is a false result), true negative (model predicted negative, which is a true result), and false negative (model predicted negative, which is a false result). From the values of the matrix, we can calculate precision, recall, and weighted F1 scores to understand the relevance and model performance.

Evaluation metrics should be selected based on the task performed by a model. It is always better to check multiple metrics to evaluate a model since a model may perform differently in different matrics. The multiclass classification case study shown in Chapter 7 has almost 50% test and 50% training data. Therefore, for a simpler evaluation, we showed only the training loss and accuracy metrics for that problem.

CHAPTER 6

Setting Up PyTorch and Google Cloud Platform Console

6.1 INTRODUCTION

In this chapter, we will discuss a method of setting up PyTorch and Google Cloud Platform (GCP) console to run a deep learning model. Google started its cloud computing service in 2008, which supports various computing frameworks, including Python [57]. By using a free Google account, we can easily run Google's web-based user interface of GCP for computing, storage, networking, security, big data/analytics, and artificial intelligence.

At the time of writing this note, GCP provides a 12-month free trial with $300 credits to any users who set up a billing account using credit card or bank details. This pay-as-you-use billing model is very convenient and economical for someone who wants to explore deep learning models using a powerful computation engine without purchasing. Other related information and eligibility on the free trial are available at Google Cloud Free Tier site [58]. Another economical option we could use here is to choose a preemptible instance for computing engine, which is much cheaper than using a regular instance. We are going to discuss the preemptible instance later in this chapter.

6.2 SETTING UP A GCP ACCOUNT

Setting up a free GCP account is very simple after creating an account through https://cloud.google.com/. If someone wants to use an existing Google account, he or she just needs to sign in (Fig. 6.1). Before account creation, it is recommended to read their quick start page available at https://cloud.google.com/gcp/getting-started#quick-starts.

After signup for a new account or log in by using an existing account, go over the Terms of Service and then select continue (Fig. 6.2).

Next, select the individual account type from the drop-down list, then fill out the address and billing information (Figs. 6.3 and 6.4). After adding a payment method, you will see the welcome message and will receive a confirmation email, similar to Fig. 6.5. Make sure to finish

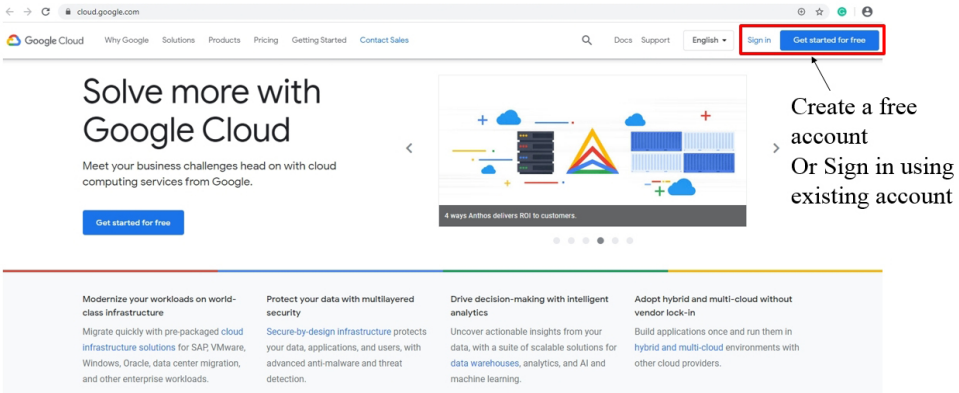

Figure 6.1: Create a free account in Google Cloud Platform or Sign in using your existing account.

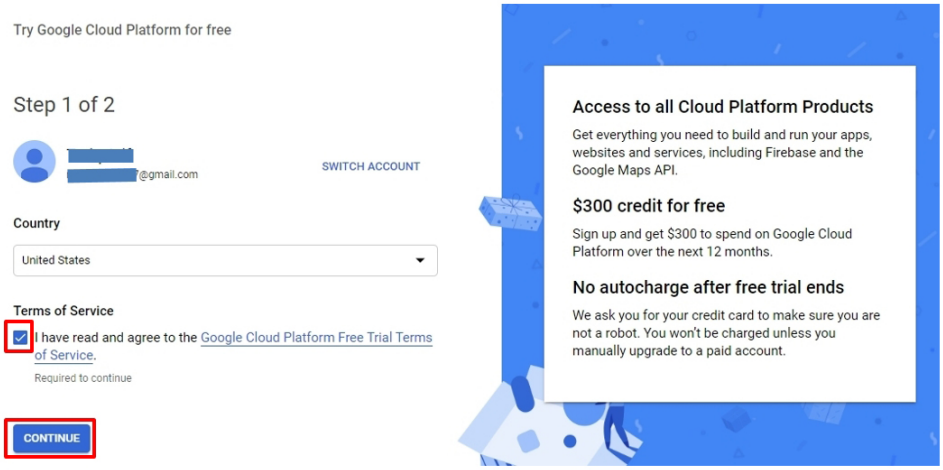

Figure 6.2: Check on Terms of Service and select Continue.

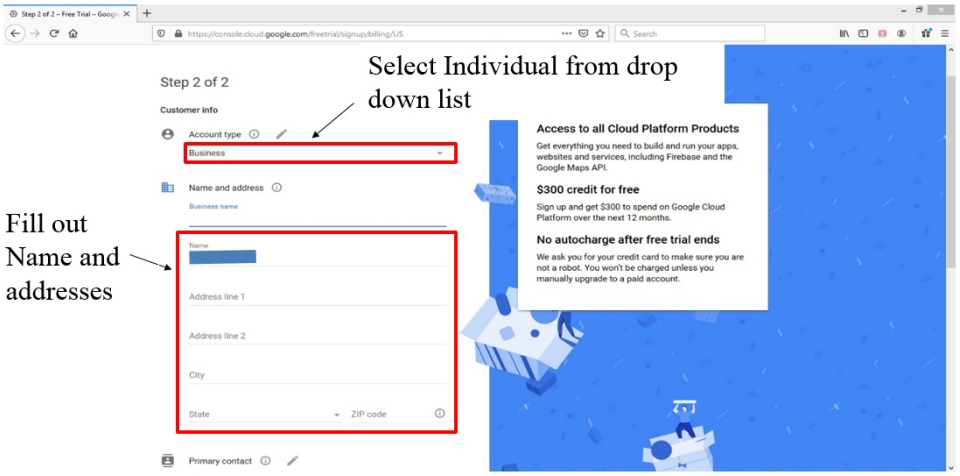

Figure 6.3: Select the individual from the drop-down menu and fill out the name and address.

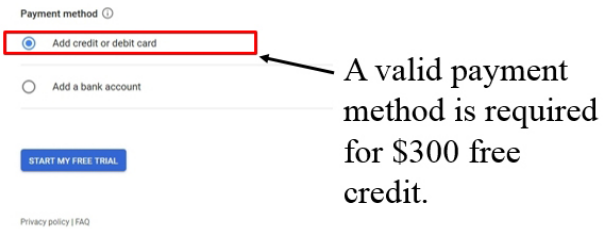

Figure 6.4: Add a payment method.

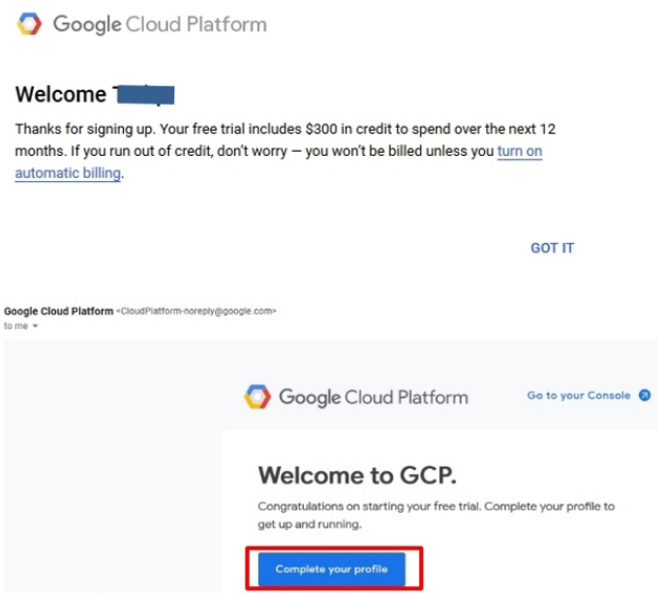

Figure 6.5: After account creation, you will receive a welcome message. Check your email and complete your profile.

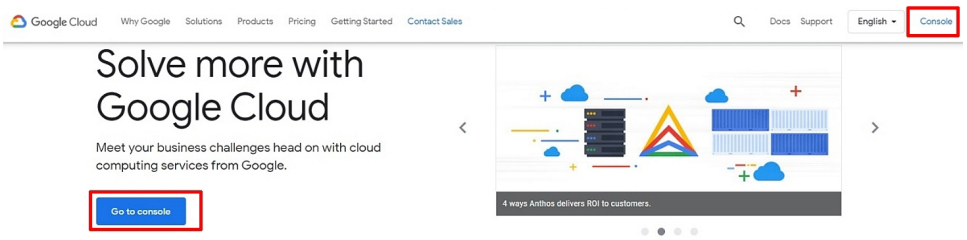

Figure 6.6: Select "go to console" or "console" to enter into GCP.

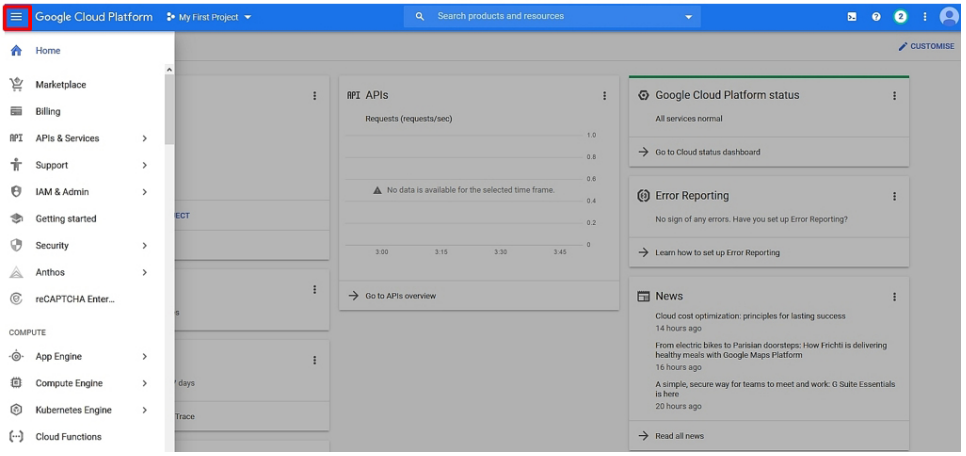

Figure 6.7: Navigation menu of GCP interface.

the "complete your profile" section from the welcome email. Next time, when you log in to the GCP, you will only need to select "go to console" or "console," as shown in Fig. 6.6.

Inside the GCP interface, a dashboard contains Project info, Compute engine, Google Cloud Platform status, Resources, API, Billing, Documentations, and other options. The navigation menu from the upper left corner (Fig. 6.7) will show a drop-down panel where you will be able to select a few different options. It is recommended to check the billing section when you run a compute engine. In this way, you will get an idea of billing and will be able to keep track of your remaining credits.

6.3 CREATE A NEW PROJECT

Now, it is time to create our first project on this cloud platform. Click on the "My First Project" and then "New Project" (Fig. 6.8). Give a project name, for example, DeepLearning1, and then select create (Fig. 6.9).

Using a free account, you may create up to 25 projects. In the figure, a message showing that 23 projects left in our quota. After finishing this step, you will see a notification on the top right corner that your project DeepLearning1 has been created.

6.4 SET UP A VM INSTANCE

To use the computing engine, you need to set up a Virtual Machine (VM) instance hosted on Google's infrastructure. Google will also let you customize the number of virtual CPUs, memory, and machine types [59]. From the navigation menu, go to the Compute Engine and then select VM instances, as shown in Fig. 6.10.

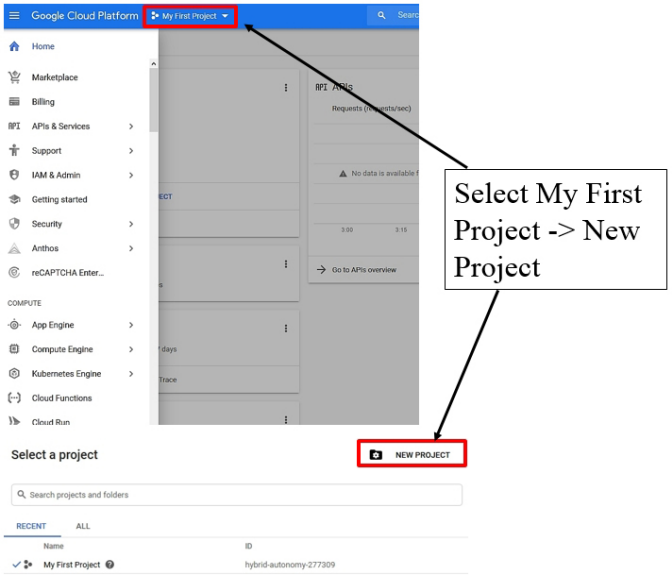

Figure 6.8: Creating a new project inside GCP.

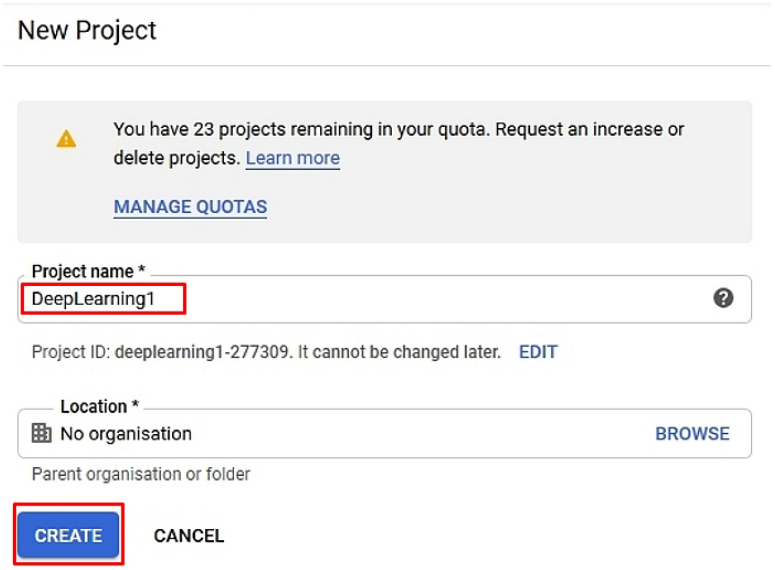

Figure 6.9: Creating a new project inside GCP.

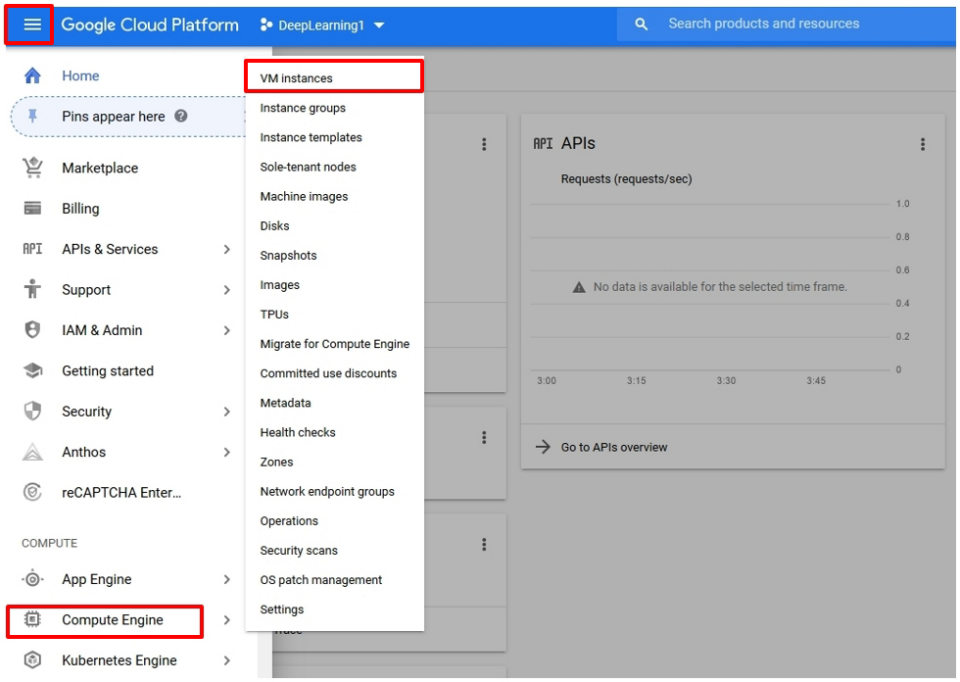

Figure 6.10: Steps for Setting up a VM instance inside GCP.

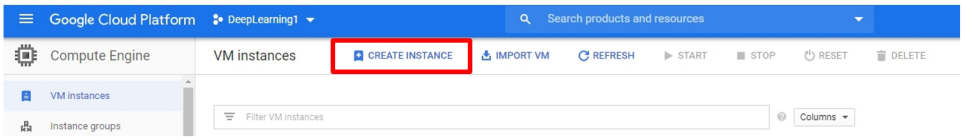

Figure 6.11: Create a new compute engine instance.

Next, select "CREATE INSTANCE." If you are creating an instance for the first time, you may see a different interface than the one shown in Fig. 6.11. You will just need to select the create instance option.

Next, assign a name for the instance (e.g., deeplearning), select a region for VM from the region drop-down menu. In this example, we have chosen us-west-1 (Oregon), but you may choose a different area. Please note that you might need to try another zone if the selected region does not have enough resources available to fulfill your request at the time of creation. Then select a custom machine type, 8 vCPU core, and 30 GB memory (Fig. 6.12).

This is just an example, and you may choose a different custom VM based on your requirement. You may use CPU only for your computation if you think CPU is good enough for handling your machine learning model. Here we intend to show you the way of selecting

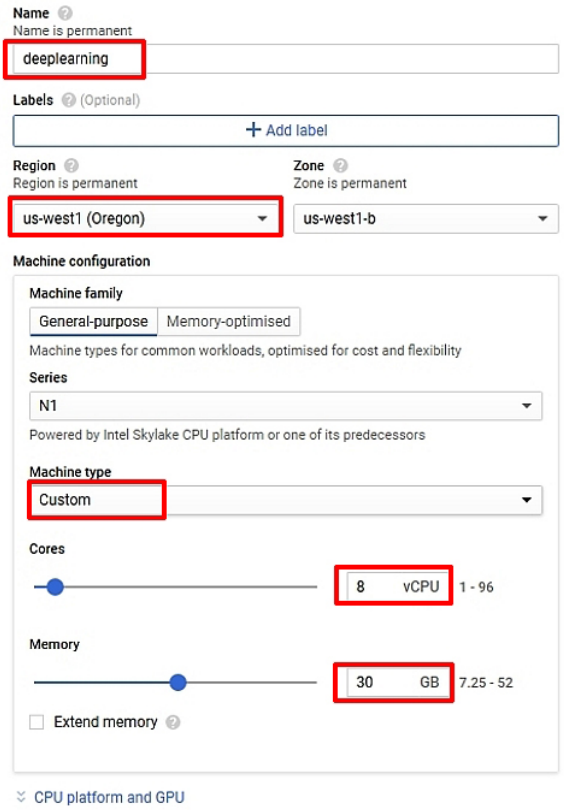

Figure 6.12: Steps for customizing cloud CPU.

GPU for your project since most of the deep learning models require a GPU for parallel computational power. If you expand the CPU platform and GPU, you will see the type of GPUs available in that location. But you won't be able to add a GPU before quota approval. You will need to request for GPU quotas, a step which we are going to show in the following section.

Next, scroll down and click change in the boot disk section. From the drop-down menu of the operating system, select Ubuntu, and its version. If you are familiar with any other operating system, you may select that one and increase the boot disk size as needed (e.g., 50 GB) for your project (Fig. 6.13).

From the firewall check on Allow HTTP traffic and Allow HTTPS traffic. After that, expand the "Management, security, disks, networking, sole tenancy" option. Go to the Management tab and copy or write the script from Fig. 6.14 for the Custom metadata, "Startup script" section. This operation will help you to perform some software installation and update every time you boot up your instance (Fig. 6.15).

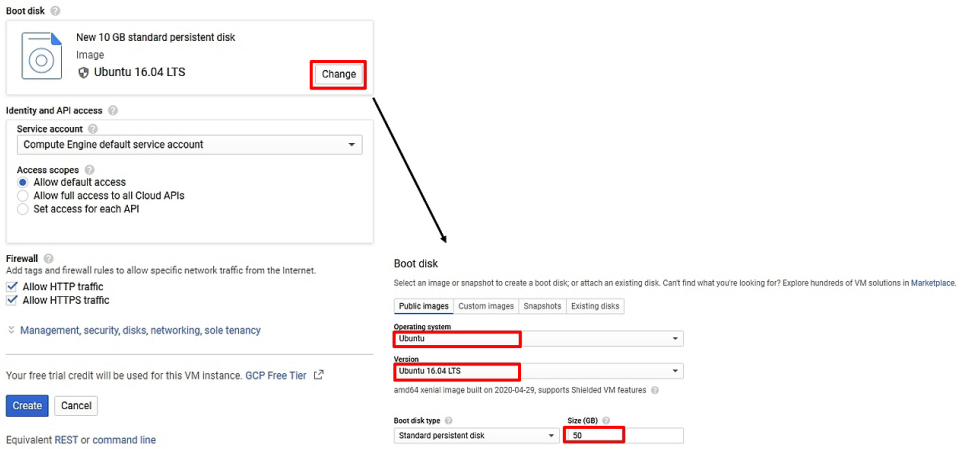

Figure 6.13: Steps for setting up a boot disk for VM.

```
#!/bin/bash
echo "Checking for CUDA and installing."
# Check for CUDA and try to install.
if ! dpkg-query -W cuda-9-0; then
  # The 16.04 installer works with 16.10.
  curl -O http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64/cuda-repo-ubuntu1604_9.0.176-1_amd64.deb
  dpkg -i ./cuda-repo-ubuntu1604_9.0.176-1_amd64.deb
  apt-key adv --fetch-keys http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64/7fa2af80.pub
  apt-get update
  apt-get install cuda-9-0 -y
fi
# Enable persistence mode
nvidia-smi -pm 1
```

Figure 6.14: Startup script for CUDA installation.

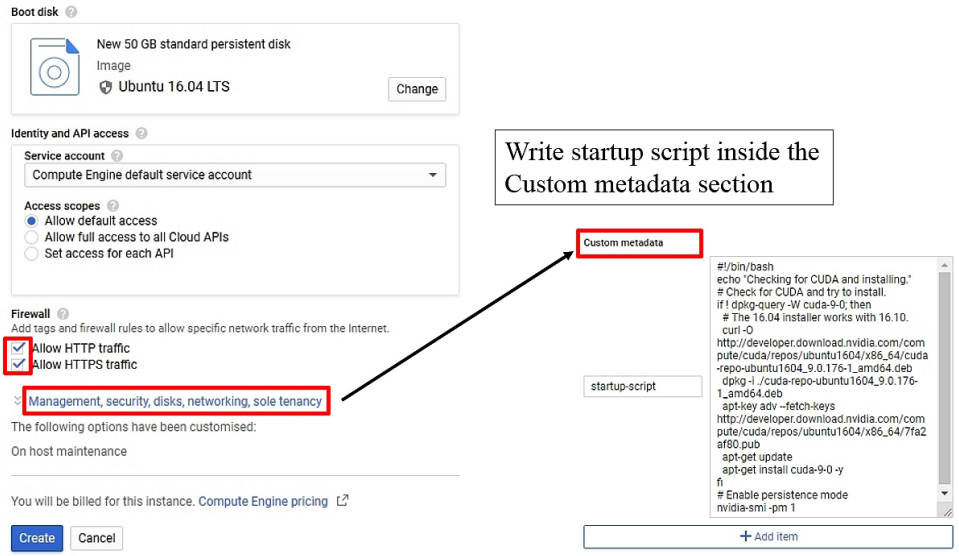

Figure 6.15: Write startup script inside Custom metadata, "Startup script" section.

Then go to the "Disks" tab and check off "Delete boot disk when instance is deleted." Finally, click on "Create" to create the computing engine. These steps are shown in Fig. 6.16.

You might see a warning message saying that the zone does not have enough resources available to fulfill the request. Very typically, if you don't select any GPU, you will not see this error message. But if you see an error message, you should try a different zone to create your instance. For example, you may change from us-west1-b to a, c or us-central1, us-east1, etc.

When the instance is successfully created, you will see a notification. At this stage, you will need to stop the instance from Compute Engine → VM instances. Select three dots, as shown in Fig 6.17, and then select stop. This trick will save some unnecessary billing in your account. You may only start the instance when you run a model or install dependencies.

6.5 GPU QUOTA REQUEST

The GPU quota request is an important step if you plan to access GPUs in Google's remote server. You may finish it before creating an instance. Go to Navigation Menu, IAM & Admin, and then select Quotas (Fig. 6.18).

If you are a new user, you might see an upgrade option on the top right corner. In that case, you will need to upgrade your account by clicking the upgrade (Fig. 6.19). Otherwise, you won't be able to submit the Quota request. Even after upgrading, Google will not charge you until your free ($300) credits have run out.

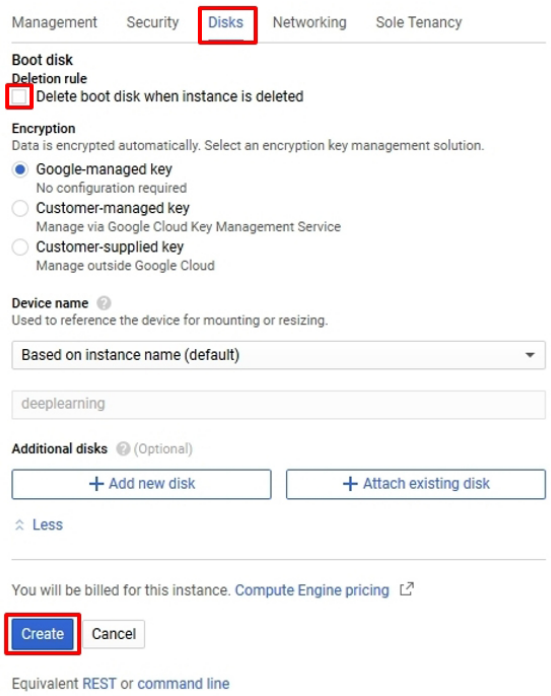

Figure 6.16: Creating the compute engine inside GCP.

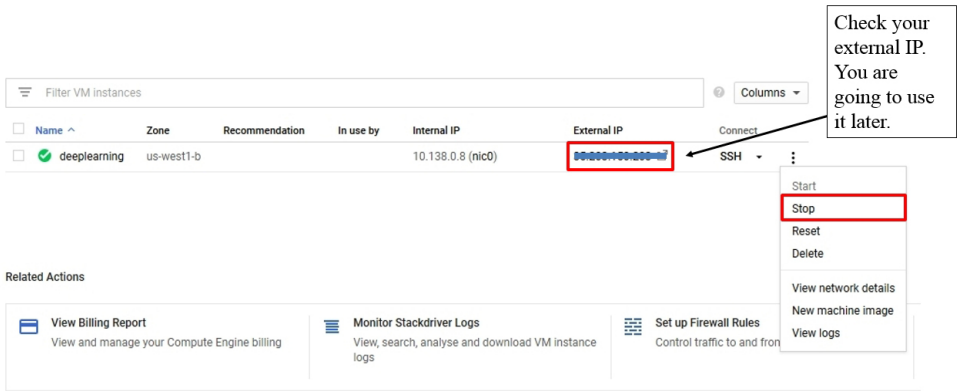

Figure 6.17: Steps for setting up a VM inside GCP.

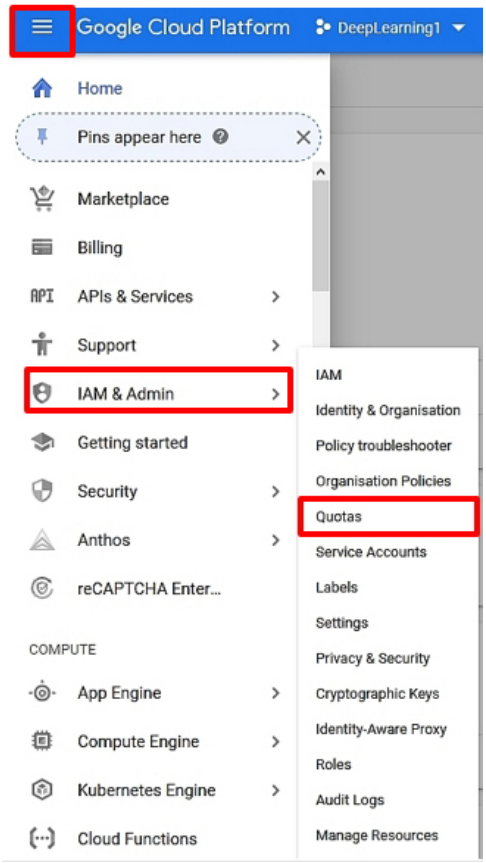

Figure 6.18: For Quota request, go to Navigation Menu, IAM & Admin, and then select Quotas.

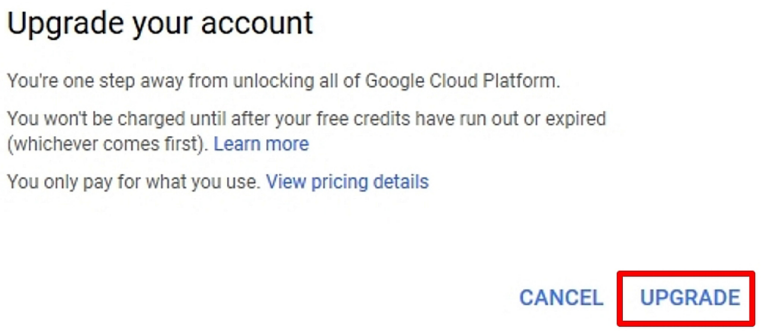

Figure 6.19: Upgrading account for Quota request.

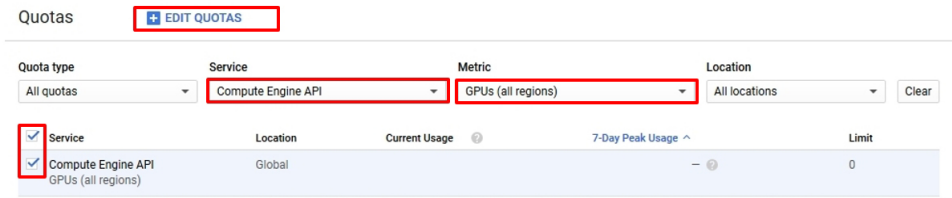

Figure 6.20: Request for GPUs in all regions.

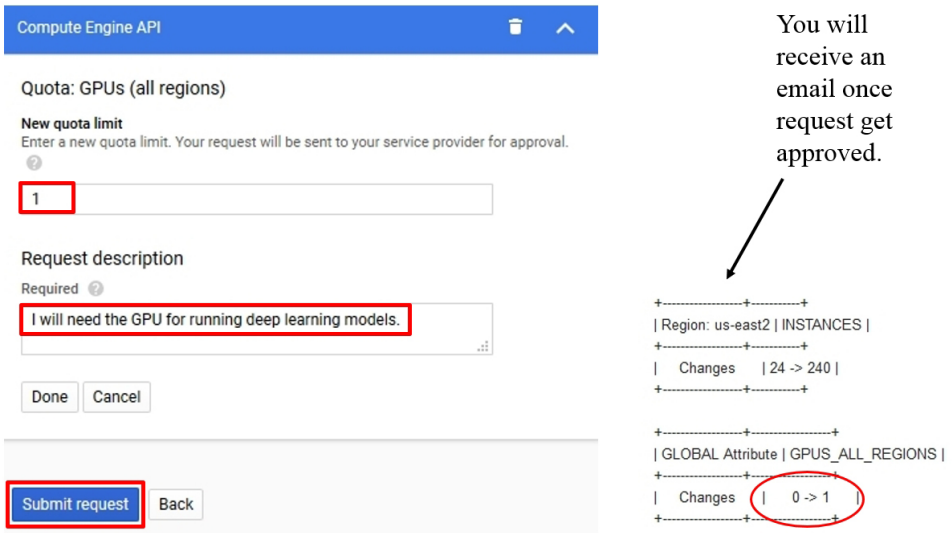

Figure 6.21: Steps for GPU Quota request submission in GCP.

Next, from the Quotas page, select Quota type → All quotas, Service Compute Engine API, Metric GPUs (all regions), and Locations All locations (Fig. 6.20).

These selections are very important. You may also request for GPU in a specific zone. Next, request for quotas by selecting EDIT QUOTAS (Fig. 6.21). Write the Number of GPU you want and provide a reason for your request in the request description section before submitting your request. You will receive an email confirmation and another email reporting your approval status within one hour to two days. It is not guaranteed that you will get the quota you requested. In that case, Google will provide you a reason for denial, and you need to try again later.

Once your quota gets approved, you will need to update your instance by adding a GPU. If you want to make any changes in your current instance, make sure to stop the running instance (deeplearning1) first, and then select the instance. You will find an EDIT button for making changes in your instance.

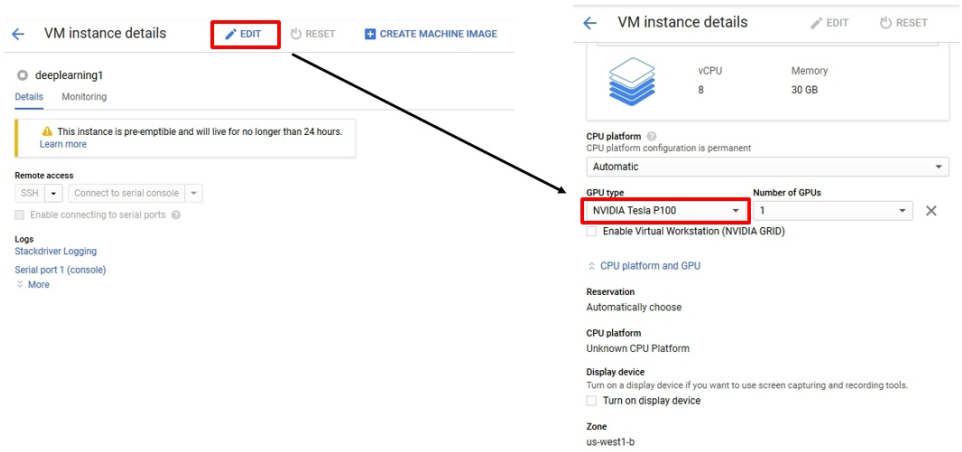

Figure 6.22: Edit VM instance and add GPU.

Select EDIT and then expand the "CPU platform and GPU" section. Here you will be able to add the number and the types of GPU you want. If you have only one GPU quota, you will not be able to add more than one GPU. If you think that you will need multiple GPUs for your project, then you should request the number of Quotas in the previous section (Fig. 6.21). You are allowed to have more than one GPU quotas. In this demonstration, we have selected one NVIDIA Tesla P100 GPU, as shown in Fig. 6.22.

If you don't find your desired GPU in the drop-down list, you will need to try a different zone. As long as you have GPU quotas, you can create compute engine instances using GPUs.

6.5.1 A COST-EFFECTIVE APPROACH

There is a cost-effective option in GCP that you may consider for your project, known as pre-emptible instance. If you select a preemptible instance, you will be able to run an instance at a much lower price than a regular instance. But the server might terminate (preempt) these instances if it needs to access these resources for other tasks. This option is a little inconvenient to use, knowing that your instance may shut down any time, but it is economical. Another notable limitation of the preemptible instance is that the compute engine is always going to terminate after they run for 24 hours. So, this is the trade-off between preemptible and non-preemptible, and you have to make your decision based on your requirements. For a new user who is running small deep learning models and still exploring different functionalities of GCP, preemptible is a practical and economical option. Note that some of the zones may not have preemptible instances available. You will find useful information related to preemptible instances in the GCPs website [60]. In this text, the model we are going to use will take about an hour to run on a GPU based compute engine. So, we have selected the "preemptible" option for our instance.

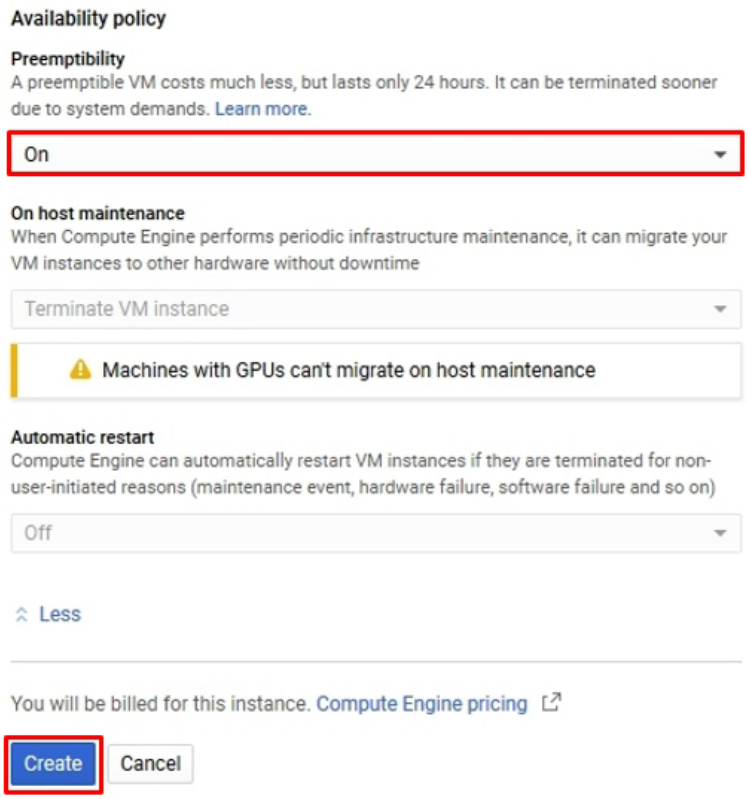

Figure 6.23: Select Preemptibility "On" and "Create" to run VM at a lower cost.

Set Preemptiblity on, under the Available policy option and Create. You will see a notification after the successful creation of your instance.

If you create your instance from the beginning after quota request approval or make another instance, then go over the previous section, the only difference is you have to select GPUs shown in this section. For any reason, if your instance does not create successfully, try to pick a different GPU or zones, or try again later. Once your GPU instance is created, you are ready to run your machine learning model in GCP.

6.6 VPC NETWORK

6.6.1 SET UP EXTERNAL IP ADDRESS

It is convenient to set up the IP address to static so that it doesn't change. The default setup for GCP is dynamic. To do this step, go to the navigation menu, VPC network, and then Ex-

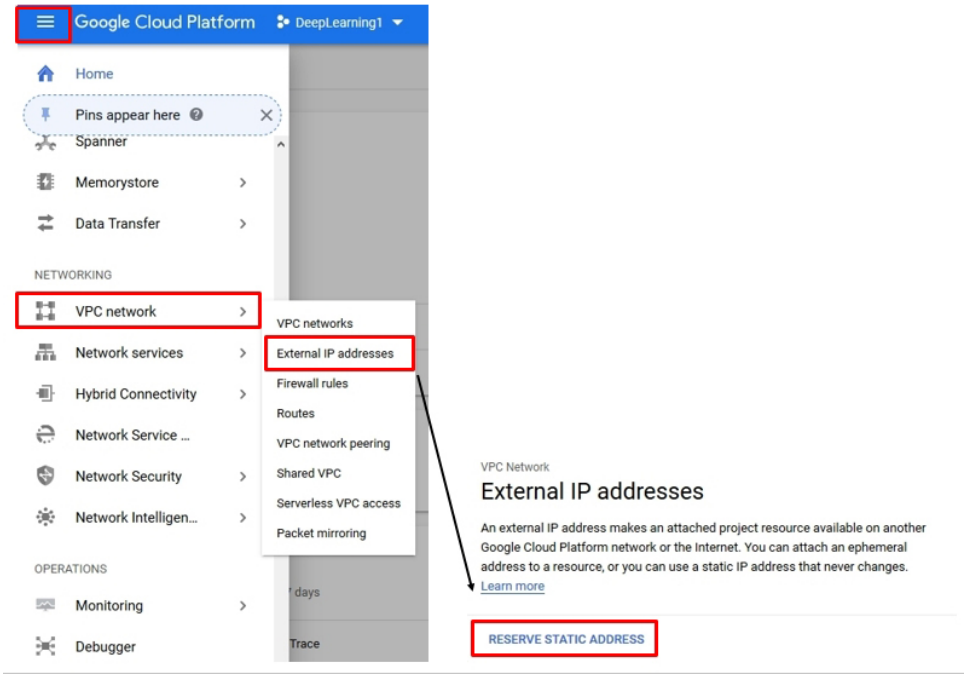

Figure 6.24: Steps for setting up an external IP address.

ternal IP addresses. If you have an existing IP address, you will find the "RESERVE STATIC ADDRESS" in the top-middle area. Otherwise check the Fig. 6.24.

You may give a name and description to the reserve static address. Also, check Network Service Tier, IP version, type, and region, as shown in Fig. 6.25. Make sure to attach the IP address from the drop-down menu to the instance you created. In our example, it is "deeplearning1."

Finally, select "RESERVE" to reserve the IP address.

6.6.2 CREATE FIREWALL RULES

To create the Firewall Rules, go to VPC Network → Firewall Rules and select CREATE FIREWALL RULE, as shown in Fig. 6.26.

Next, select Targets as All instances in the network and type a source IP ranges (0.0.0.0/0). Select specified protocols and ports. Then check on transmission control protocol (tcp), and give a four- or five-digit port number (e.g., 8081). Check this setting carefully and click create (Fig. 6.27).

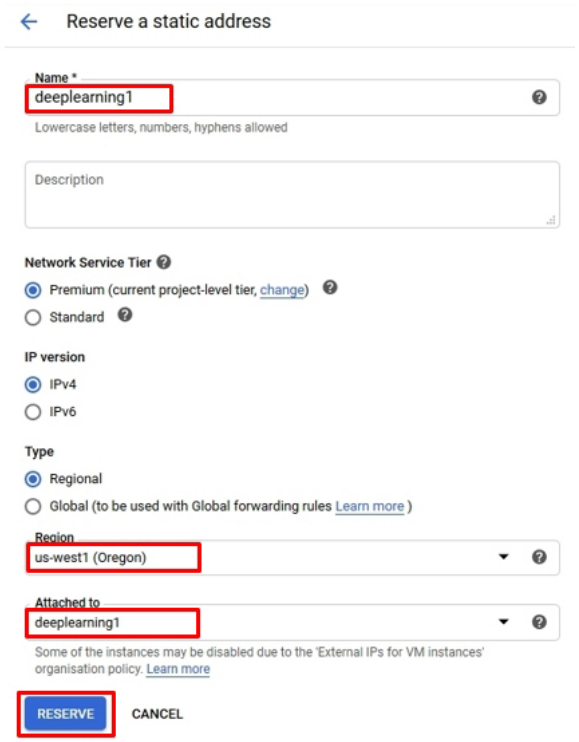

Figure 6.25: Steps for reserving a static IP address.

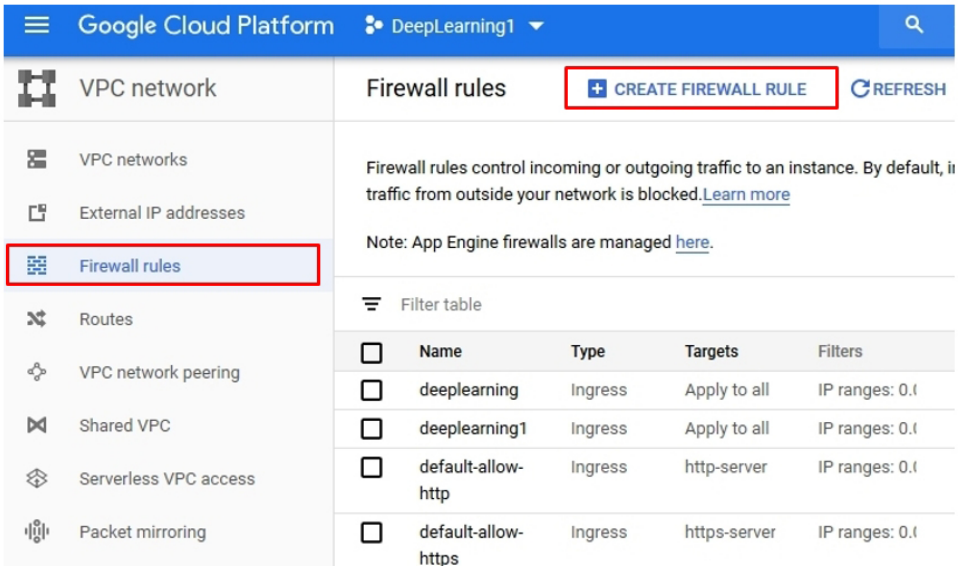

Figure 6.26: Steps for creating Firewall rules in GCP.

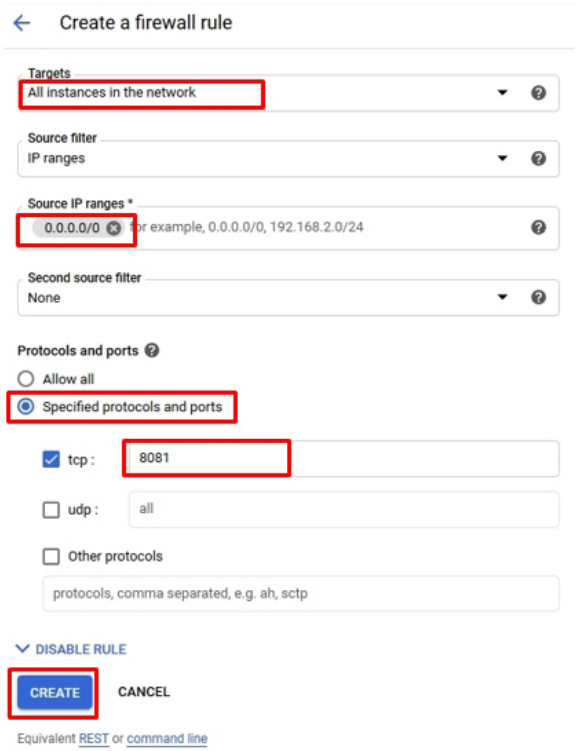

Figure 6.27: Steps for creating Firewall rules in GCP and specify a port number.

Figure 6.28: To install Anaconda on VM instance, select SSH.

```
wget http://repo.continuum.io/archive/Anaconda3-5.3.1-Linux-x86_64.sh
```

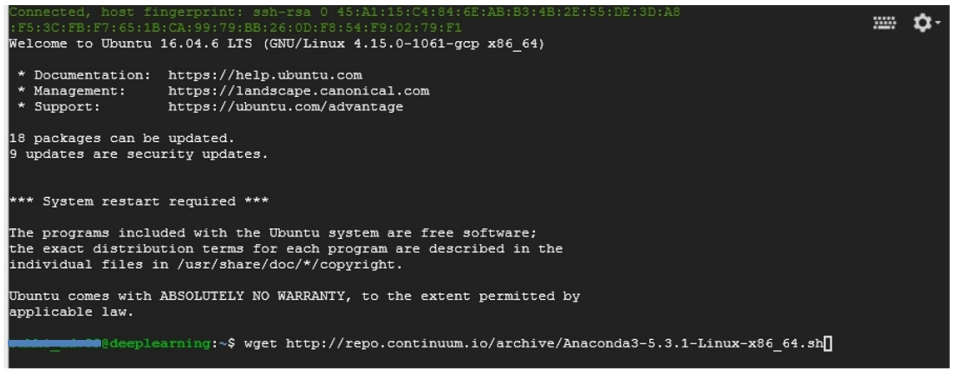

Figure 6.29: Steps for Anaconda download and installation in GCP.

6.7 SETTING UP VM INSTANCE TO RUN MODELS

For our deep learning project, we are going to install Anaconda on GCP. It has many amazing tools, web applications, and Python libraries that are useful for our project. The Jupyter Notebook live code editor is also available in Anaconda. So, in this section, we are going to show how to install Anaconda on the virtual machine, set up Jupyter Notebook, and other dependencies such as Pytorch and OpenCV.

6.7.1 ANACONDA INSTALLATION ON VM

To install Anaconda in our VM instance, first, we will need to start the VM instance from Navigation Menu → Compute Engine → VM instance. Click on the SSH to access the VM through Ubuntu platform that we selected (Fig. 6.28).

This selection will bring us to the following shell window. Then type the command shown in Fig. 6.29, and press enter. Note that the site used in the command line is http://repo.continuum.io. Here we might use https://repo.anaconda.com/ site for installing Anaconda. We may also change the Anaconda version if needed.

To access the bash shell-environment type the following command and press enter. Review the license agreement and keep pressing enter to continue (Fig. 6.30).

```
bash Anaconda3-5.3.1-Linux-x86_64.sh
```

```
Anaconda3-5.3.1-Linux-x86_6 100%[=======================================>] 637.03M   200MB/s    in 3.2s

2020-05-15 23:59:10 (198 MB/s) - 'Anaconda3-5.3.1-Linux-x86_64.sh' saved [667976437/667976437]

[redacted]@deeplearning:~$ bash Anaconda3-5.3.1-Linux-x86_64.sh
```

```
[redacted]@deeplearning:~$ bash Anaconda3-5.3.1-Linux-x86_64.sh

Welcome to Anaconda3 5.3.1

In order to continue the installation process, please review the license
agreement.
Please, press ENTER to continue
>>>
```

Figure 6.30: Steps for Anaconda installation in GCP.

```
Do you accept the license terms? [yes|no]
[no] >>>
Please answer 'yes' or 'no':'
>>> yes
```

Figure 6.31: Steps for Anaconda installation in GCP.

```
installing: scikit-learn-0.19.2-py37h4989274_0 ...
installing: astropy-3.0.4-py37h14c3975_0 ...
installing: odo-0.5.1-py37_0 ...
installing: statsmodels-0.9.0-py37h035aef0_0 ...
installing: blaze-0.11.3-py37_0 ...
installing: seaborn-0.9.0-py37_0 ...
installing: anaconda-5.3.1-py37_0 ...
installation finished.
Do you wish the installer to initialize Anaconda3
in your /home/[redacted]/.bashrc ? [yes|no]
[no] >>> yes
```

Figure 6.32: Steps for Anaconda installation in GCP.

After that, type "yes" and press "enter" to accept the license terms (Fig. 6.31).

These steps will install related libraries and packages. In the end, you will need to type "yes" and "enter" again to confirm and initialize Anaconda in your home directory (Fig. 6.32).

Next, install Microsoft Visual Studio (VS) code by typing "yes" and press "enter" again (Fig. 6.33).

After installing Microsoft VS, type the following command in Fig. 6.34 to initialize interactive shell session.

Now the VM instance is ready for running interactive sessions.

6.7.2 JUPYTER NOTEBOOK SET UP

In this section, we are going to setup Jupyter Notebook for VM instance. Enter the following commands to configure Jupyter Notebook for running it in the server (Fig. 6.35).

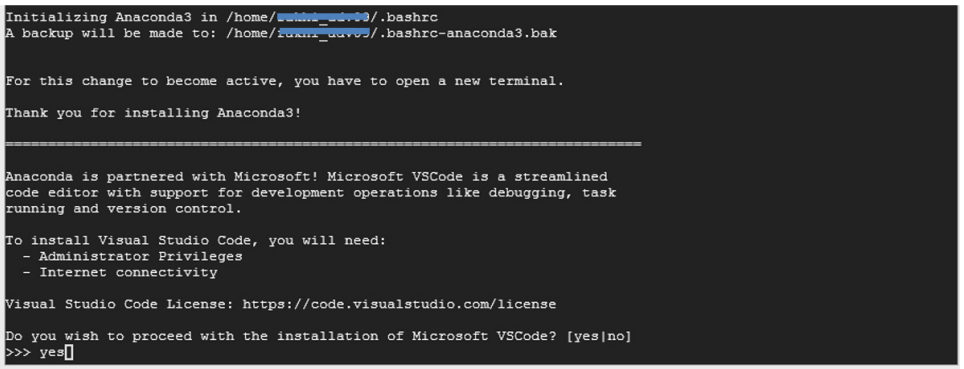

Figure 6.33: Steps for installing Microsoft Visual Studio (VS).

```
source ~/.bashrc
```

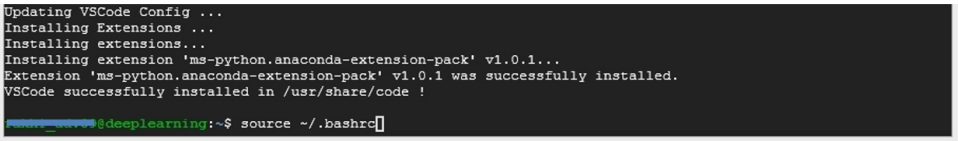

Figure 6.34: Steps for initializing interactive shell session.

```
jupyter notebook --generate-config
cd ~/.jupyter/
vi jupyter_notebook_config.py
```

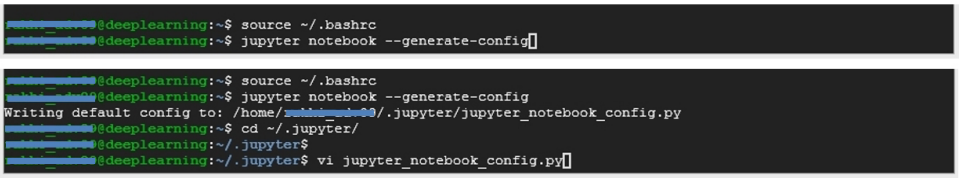

Figure 6.35: Steps for configuring Jupyter Notebook in GCP.

This command will open the Jupyter Notebook configuration file for VM instance. Press "i" to edit this configuration file and add the lines shown in Fig. 6.36. The port number used in this example will be the one that you specified during Firewall settings (Fig. 6.27).

After editing the configuration file, press "Esc" and type ":wq" in your keyboard to save and exit (Fig. 6.37). If you are interested in setting up a password now, check the Jupyter Notebook configuration for public server [61]. You may close your session now.

Now you are ready to launch Jupyter Notebook and execute code in the server. Next, go to the SSH terminal and type terminal multiplexer command "tmux" (Fig. 6.38).

```
c = get_config()
c.NotebookApp.ip = '*'
c.NotebookApp.open_browser = False
c.NotebookApp.port = 8081
```

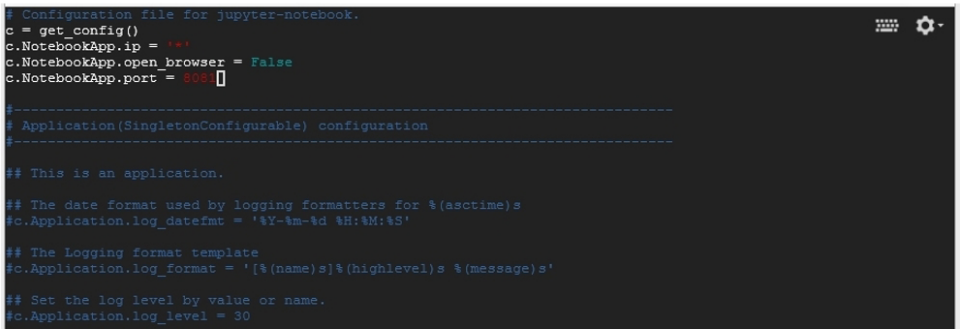

Figure 6.36: Steps for configuring Jupyter Notebook in GCP.

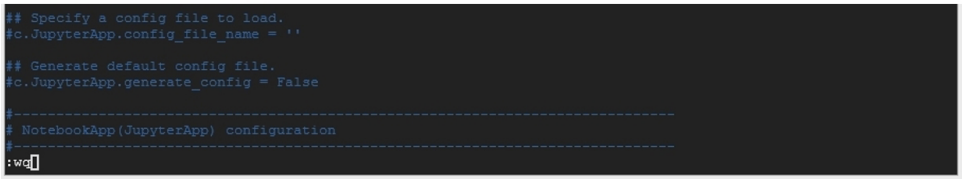

Figure 6.37: Press "Esc" and type ":wq" in your keyboard to save and exit.

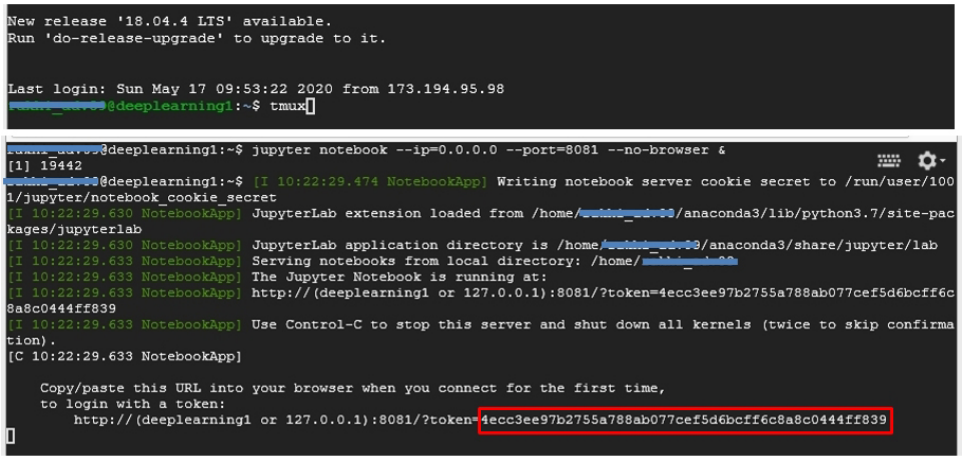

Figure 6.38: Steps for running Jupyter Notebook GCP.

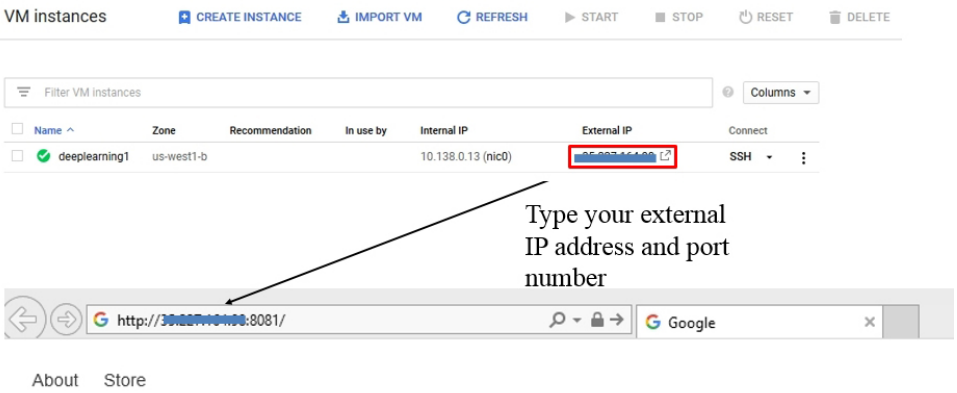

Figure 6.39: Enter `http://<External Static IP Address>:<Port Number>` in a web browser.

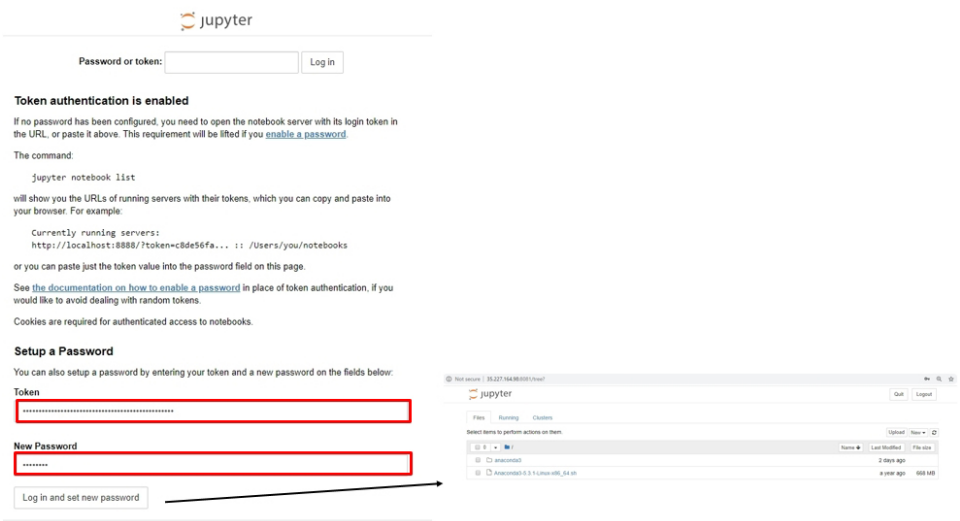

Figure 6.40: Copy the login token shown in Fig. 6.38 and enter it to the token section and then create a new password.

After that, open your web browser and type following:

`http://<External Static IP Address>:<Port Number>`

(in our example it is, http://**.**.***.***:8081), as shown in Fig. 6.39.

```
tmux
jupyter notebook --ip=0.0.0.0 --port=8081 --no-browser &
```

Figure 6.41: Enter commands in the SSH terminal before accessing Jupyter Notebook through a webpage.

If you are running the Jupyter Notebook for the first time, you have to copy the login token shown in Fig. 6.38 (red box) and Enter it to the browser's token section, as shown in Fig. 6.40. Then you can create a new password.

If your token doesn't work, check the message in the shell window. You probably ended up opening multiple ports. You may restart the VM instance and use the new token to create a password. After the successful installation of the Jupyter Notebook in your GCP, you will be able to write or edit a script and run it in the VM instance through this interface.

Next time when you run the SSH terminal (shown in Fig. 6.28), you will only need to enter the command lines shown in Fig. 6.41.

After that, enter the url (http://**.**.***.***:8081) for your server in the web browser. The browser will ask for the password that you created (Fig. 6.40), and then the Jupyter Notebook interface will be opened.

CHAPTER 7

Case Study: Practical Implementation Through Transfer Learning

7.1 PROBLEM STATEMENT

In this chapter, to demonstrate the use of deep learning, we have used a transfer learning process for a multiclass image classification problem. Our goal is to implement a deep learning model to determine the Make, Model, and Year of a car. In order to train and test our model, we have used an organized car dataset containing a total of 16,185 different car images. This dataset (Stanford's Cars Dataset) was used in a Kaggel's competition and can be downloaded from [62]. An earlier version of the dataset was also used in research for 3D object representation of fine-grained categorization of rigid classes [63].

In the Stanford's Cars Dataset, among 16,185 car images, 196 different classes (Make, Model, and Year) of cars are divided into training images (8,144) and testing images (8,041). Inside the car_devkit zip file, data classes, and corresponding bounding boxes to crop the images are available. From their website, the cars_test_annos_withlabels.mat file was used to find out our test data set accuracy. Note that there is a similar file (cars_test_annos.mat) available on their site that doesn't have the classes defined in it. So, we recommend everyone to check Stanford Cars Data set carefully before working on this project.

To identify car make and model from an image, we need to train a deep learning model using the training data and then test the performance of the model by using the testing data. A pre-trained deep neural network, known as Efficient B7, is used for this task to extract features from the training dataset.

A schematic of the transfer learning process using the pre-trained network is shown in Fig. 7.1. Finally, for testing purposes, we are going to feed several random car pictures (downloaded from the internet) to see how well the new deep learning model performs outside of the given dataset.

7.2 DATA PROCESSING

The data processing step is very critical for using any deep learning model effectively. We have to carefully check the labels and names for the images and the consistency between training

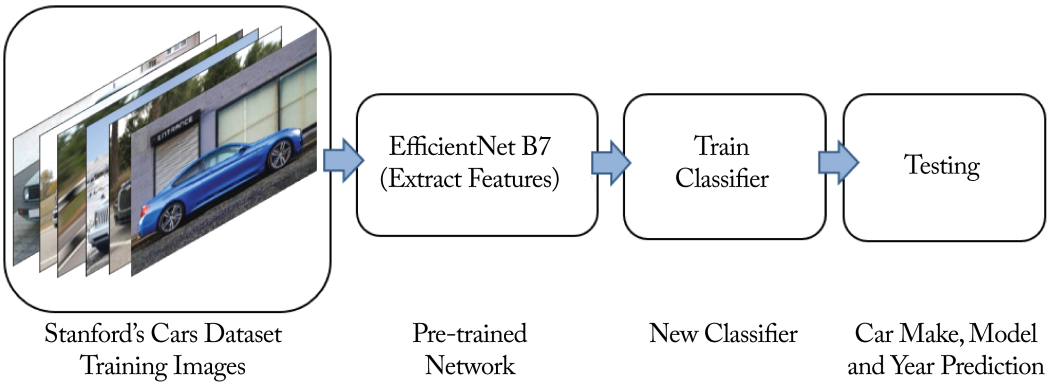

Figure 7.1: Schematic of transfer learning using a pre-trained deep neural network, EfficientNet B7.

and testing data. A preprocessing step will help us to call a large volume of images using simple Python codes.

For this case study, we made a folder called "input," which contains all the relevant files required to run our model. If you want to reproduce the result, please check the Fig. 7.2 containing the screenshot of our input folder and file names.

Please note that we converted ".xlsx" files to ".csv" files, and a new column is added to the "cars_test_annos.csv" that shows classes of each test image copied from "cars_test_annos_withlabels.mat." We made this change to check the accuracy of our deep learning model on the testing dataset.

7.3 UPLOAD DATA INTO STORAGE BUCKET

To upload the data folder "input" in the cloud console, follow the steps shown below.

Step 1: Go to GCP Navigation Menu → File Storage → CREATE BUCKET

Step 2: Choose a bucket name (e.g., deeplearning_bucket1) and keep on selecting "continue" to accept all the default settings (Figs. 7.4, 7.5, and 7.6).

Step 3: Select "CREATE" to create the bucket folder (Fig. 7.6).

Step 4: Select "Upload folder" from the bucket and upload the "input" folder (Fig. 7.7).

7.4 TRANSFERRING FILE TO VM INSTANCE

Next, we are going to transfer "input" files from the bucket to our VM instance. To do this transfer process, we have to start the compute engine and establish SSH connection. Follow the instructions given below and check Fig. 7.8.

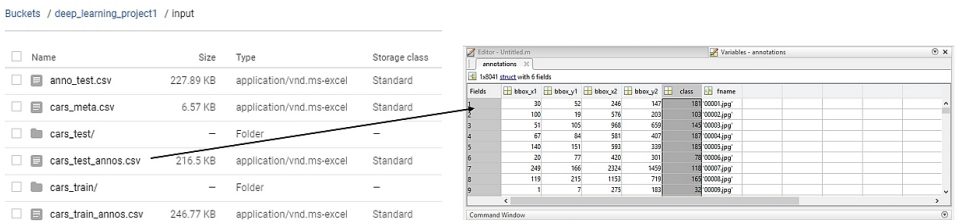

Figure 7.2: Screenshot of "input" folder, and its files.

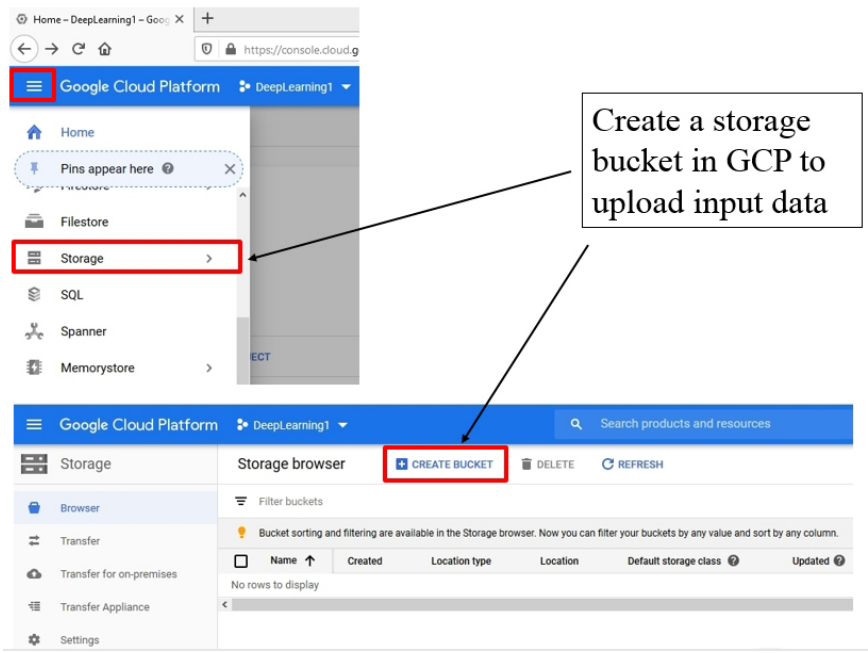

Figure 7.3: Steps for creating a storage bucket in GCP.

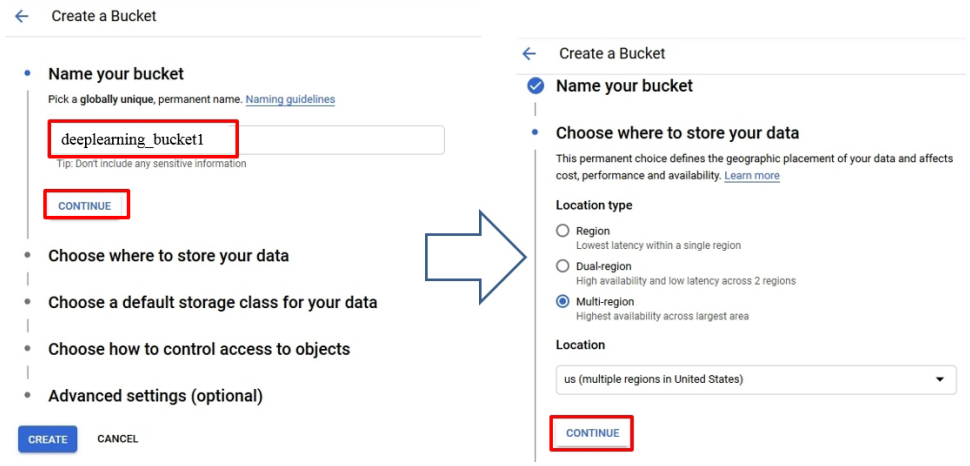

Figure 7.4: Steps for creating a storage bucket in GCP.

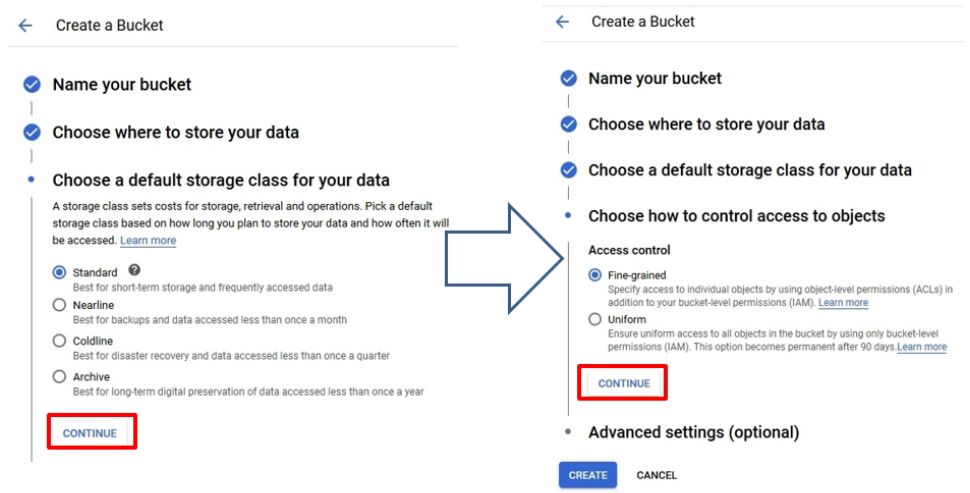

Figure 7.5: Steps for creating a storage bucket in GCP.

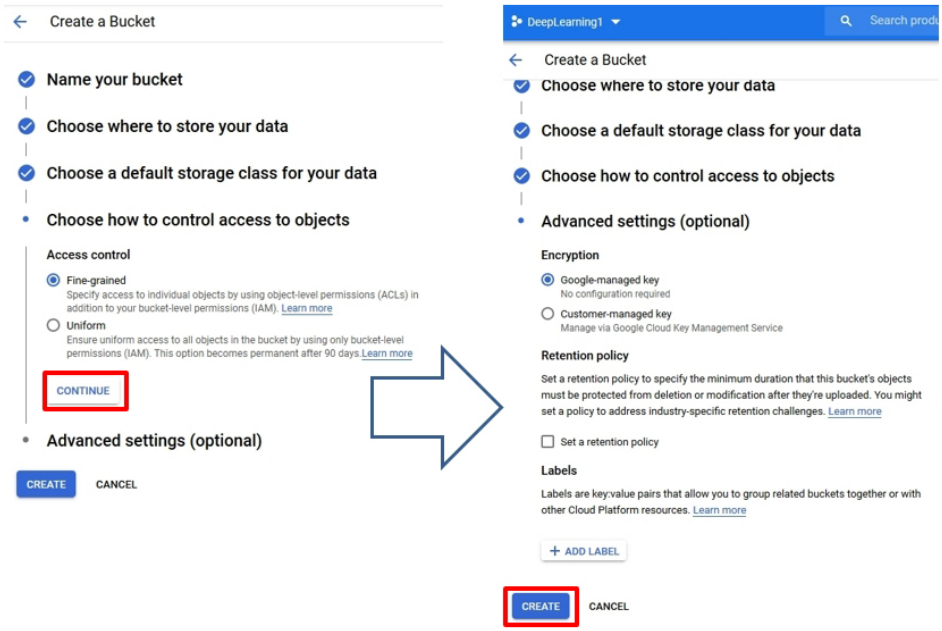

Figure 7.6: Steps for creating a storage bucket in GCP.

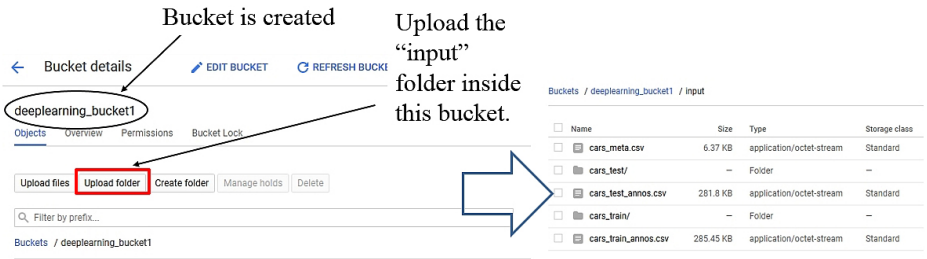

Figure 7.7: Steps for uploading files in the GCP bucket.

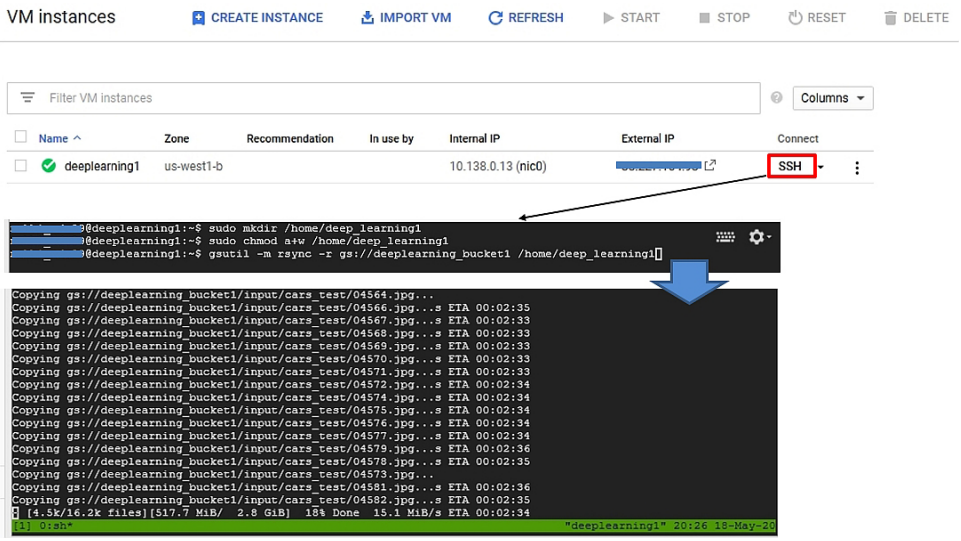

Figure 7.8: Steps for transferring "input" files to VM instance.

Step 1: Start the VM instance and select "SSH" terminal.

Step 2: Create a directory (e.g., deep_learling1) inside the home of instance. In the command line type "sudo mkdir /home/deep_learning1" and press "enter."

Step 3: Give permission to read and copy files inside the deep_learning1 folder. In the command line type "sudo chmod a+w /home/deep_learning1" and press "enter."

Step 4: Copy everything from deeplearning_bucket1 to VM instance folder (deep_learning1). In the command line type "gsutil -m rsync -r gs://deeplearning_bucket1 /home/deep_learning1" and press "enter."

The "rsync" command is used to automate the synchronization of a local file system directory. You will notice in the command window that all the files and folders of "input" are transferring to VM instance (Fig. 7.8).

7.5 TRANSFER LEARNING STEPS

7.5.1 DEFINE LOSS FUNCTION AND OPTIMIZER

To bundle data efficiently and access different object variables, we are going to create a Python class called "`DefaultConfigs`." The class instantiation (make a new object and initialize) is done by calling it, `config = DefaultConfigs( )`. In Fig. 7.9, the train_data and test_data are specifying the location of the training and testing images inside the home directory. We

```
class DefaultConfigs(object):
    train_data = "/home/deep_learning1/input/cars_train/" # location of training images in the bucket
    test_data = "/home/deep_learning1/input/cars_test/"   # location of testing images in the bucket
    num_classes = 196
    img_weight = 224
    img_height = 224
    channels = 3
    lr = 0.01                                          # lr = learning rate
    batch_size = 16
    epochs = 20
    seed= 112
    path = './deep_models/'

config = DefaultConfigs()
```

Figure 7.9: Make Python class object and initialize parameters.

```
!pip install --upgrade pip
!pip install six numpy scipy Pillow matplotlib scikit-image opencv-python imageio Shapely
!pip install imgaug
!pip install efficientnet_pytorch
!pip install torch torchvision #for CUDA version 10.1
!pip install torch==1.3.1+cu92 torchvision==0.4.2+cu92 -f https://download.pytorch.org/whl/torch_stable.html
    #for CUDA version 9.2
```

Figure 7.10: Install dependencies in VM instance.

have also created checkpoints to save model weights, and the model parameters that give the best accuracy are always going to be saved inside deep_models. Here, all the images have the same size (224 by 224), and 3 channels are used for red, green, and blue colors.

The number of samples that go through the model, before updating the parameters (also known as the batch size) is set as 16. If we increase the batch size, the model performance may improve, but it will take a significant amount of memories.

An optimum epoch selection that will not cause overfitting is also important. We have selected 20 epochs, i.e., forward and backward propagation will be done 20 times through the entire data set. This selection is critical because a low number of epochs might end up underfitting the model. One way to understand the fitting is to look into the training and testing accuracy. Note that in this case study, we haven't divided the training data into training and validation sets.

If the training accuracy continuously increases and testing accuracy decreases, that is an indication of overfitting the model. For this problem, the gradient descent hyperparameter (also known as the learning rate) is set to 0.01. This learning rate is a standard starting point for many deep learning models.

7.5.2 INSTALL DEPENDENCIES

If we are running our code in GCP for the first time, we will need to install some libraries using pip command (Fig. 7.10). Since we are going to use the pre-trained network (EfficientNet-B7), we will also need to load it from PyTorch.

```
import os
import sys
import json
import torch
import torchvision
import shutil
import numpy as np
import pandas as pd
import random
import warnings
import pathlib

import cv2
from PIL import Image

from torch import nn,optim
import torch.nn.functional as F
from torch.autograd import Variable
from torch.utils.data import Dataset,DataLoader
from torch.optim import lr_scheduler
from torchvision import transforms as T
from torchvision import models

from imgaug import augmenters as iaa  #image augmentation package
from collections import OrderedDict
from sklearn.model_selection import train_test_split
```

Figure 7.11: Import all required libraries.

```
if not os.path.exists(config.path):
    os.mkdir(config.path)
```

Figure 7.12: Create a directory if it does not exist.

We are going to run this block of code shown in Fig. 7.10 only once in GCP. Typically, the model needs to be run many times for tuning. Next time onward, we don't have to install these dependencies. Also, note that the "pip install torch" depends on the version of the CUDA that we installed in the startup-script.

7.5.3 IMPORT LIBRARIES

Import all the libraries that are needed to run our model. If any of these libraries looks unfamiliar, we would suggest you go through the respective library documentation.

We can also create a directory if it doesn't exist in that location. Follow the code in Fig. 7.12 for this task.

```
def save_checkpoint(state, is_best_loss):
    if is_best_loss:
        torch.save(state, config.path+'model_best.pth.tar')

def get_learning_rate(optimizer):
    lr=[]
    for param_group in optimizer.param_groups:
       lr +=[ param_group['lr'] ]

    #assert(len(lr)==1)
    lr = lr[0]

    return lr
```

Figure 7.13: Define checkpoint and a learning rate function.

7.5.4 CHECKPOINT AND ADAPTIVE LEARNING RATE

Next, we are going to create a checkpoint folder that will save the best parameters of our deep learning model. We are also going to use an adaptive learning rate, which means that the learning rate is going to decrease as the model converges. The optimizer will do this job for us, and this step will help us to avoid overshooting the minima (Fig. 7.13).

After that, we are going to define a class object that will store the average and current value only (Fig. 7.14).

7.5.5 SET SEED

We may set a seed to make this case study reproducible (Fig. 7.15). Since our model and inputs are not going to change, we may also use "`torch.backends`."

7.5.6 DATASET CLASS AND AUGMENTATION

For effective data manipulation, we are going to create Dataset class objects, as shown in Fig. 7.16. In this step, we are also going to define a function for standard normalization for each channel.

7.5.7 DATA PREPROCESSING

The next important thing is to check the dataset and do some preprocessing. Here, we may read the data using index locations. Stanford's car data set also provides the bounding box coordinates to crop the car image inside a rectangle. If we use the cropped images for training, our model accuracy will increase.

```
class AverageMeter(object):
    """Computes and stores the average and current value"""
    def __init__(self):
        self.reset()

    def reset(self):
        self.val = 0
        self.avg = 0
        self.sum = 0
        self.count = 0

    def update(self, val, n=1):
        self.val = val
        self.sum += val * n
        self.count += n
        self.avg = self.sum / self.count
```

Figure 7.14: Class object to store the average and current value.

```
random.seed(config.seed)
np.random.seed(config.seed)
torch.manual_seed(config.seed)
torch.cuda.manual_seed_all(config.seed)
os.environ["CUDA_VISIBLE_DEVICES"] = "0"
torch.backends.cudnn.benchmark = True
warnings.filterwarnings('ignore')
```

Figure 7.15: Set manual seed and use `torch.backends`.

```
# create dataset class
class CarsDataset(Dataset):
    def __init__(self,images_df,base_path,augment=True,mode="train"):
        if not isinstance(base_path, pathlib.Path):
            base_path = pathlib.Path(base_path)
        self.images_df = images_df.copy()
        self.augment = augment
        self.images_df.frame = self.images_df.frame.apply(lambda x:base_path / x)
        self.mode = mode

    def __len__(self):
        return len(self.images_df)

    def __getitem__(self,index):
        X1 = self.read_images(index)
        if not self.mode == "test":
            y = int(self.images_df.iloc[index].Id)
        else:
            y = str(self.images_df.iloc[index].frame)
        if self.augment:
            X1 = self.augmentor(X1)

        X1 = T.Compose([T.ToPILImage(),T.ToTensor(),T.Normalize([0.485, 0.456, 0.406],[0.229, 0.224, 0.225])])(X1)

        return X1.float(), y
```

Figure 7.16: Code for creating a dataset class.

```
def read_images(self,index):
    row = self.images_df.iloc[index]
    filename = str(row.frame)
    img = np.array(cv2.imread(filename,1))

    # For cropping images
    ################################
    x1 = self.images_df.iloc[index].bbox_x1
    x2 = self.images_df.iloc[index].bbox_x2
    y1 = self.images_df.iloc[index].bbox_y1
    y2 = self.images_df.iloc[index].bbox_y2
    img = img[y1:y2,x1:x2]
    ################################

    img = cv2.resize(img,(config.img_weight,config.img_height))
    return img
```

Figure 7.17: Code for reading images and cropping using bounding boxes.

In the following code, we define a "read_images" function that will use the bounding boxes given by Stanford's cars dataset and do the cropping (Fig. 7.17).

Next, we can create another function, "augmentor" to augment our training images (Fig. 7.18). Many research studies show that augmenting training images will improve the learning process. Here, we rotate the images, so that the model can view it differently from different angles and learn better.

After the image augmentation, we are going to read the training and testing CSV files using the panda library. We are also going to remove the quotation marks that exist in the

```
def augmentor(self,image):
    seq = iaa.Sequential([
        iaa.OneOf([
            iaa.Affine(rotate=90),
            iaa.Affine(rotate=180),
            iaa.Affine(rotate=270),
            iaa.Affine(shear=(-16, 16)),
            iaa.Fliplr(0.5),

        ])], random_order=True)

    image_aug = seq.augment_image(image)

    return image_aug
```

Figure 7.18: Image augmentation function to learn better from training images.

```
train_files = pd.read_csv("/home/deep_learning1/input/cars_train_annos.csv")
test_files = pd.read_csv("/home/deep_learning1/input/cars_test_annos.csv")

# to remove the quotation marks from cars_train_annos.csv image name
train_files['frame'] = [i[1:-1] for i in train_files['frame']]
test_files['frame'] = [i[1:-1] for i in test_files['frame']]

# Rename the class lable to ID for consistency between cars_train_annos.csv and anno_test.csv
#columns= ['unnamed:0','x1','y1','x2','y2','Id','frame']
columns = train_files.columns.tolist()
columns.remove('class')
columns.insert(5,'Id')
train_files.columns = [i for i in columns]

## test files column correction
# columns1= ['unnamed:0','x1','y1','x2','y2','frame','Id']

# Make the labels form 0 to n-1 for n classes to use CrossEntropyLoss()
train_files['Id'] = [(i-1) for i in train_files['Id']]
test_files['Id'] = [(i-1) for i in test_files['Id']]
```

Figure 7.19: Read training and testing CSV files, and make consistent column heading names.

original file. This step is required for the model to access the correct classes and bounding box information of the corresponding images (Fig. 7.19).

After executing up to the previous code (Fig. 7.19), we may check our test and train file heads using the "head" function. It will give us the result shown in Fig. 7.20. As we can see, the quotation mark from the image name has been removed.

Now, the files can be accessed easily by the deep learning model using the scripts below. In Fig. 7.21, the train_loader and val_loader load images for training and testing, respectively.

```
test_files.head()
```

	Unnamed: 0	bbox_x1	bbox_y1	bbox_x2	bbox_y2	frame	Id
0	1	30	52	246	147	00001.jpg	180
1	2	100	19	576	203	00002.jpg	102
2	3	51	105	968	659	00003.jpg	144
3	4	67	84	581	407	00004.jpg	186
4	5	140	151	593	339	00005.jpg	184

```
train_files.head()
```

	Unnamed: 0	bbox_x1	bbox_y1	bbox_x2	bbox_y2	Id	frame
0	1	39	116	569	375	13	00001.jpg
1	2	36	116	868	587	2	00002.jpg
2	3	85	109	601	381	90	00003.jpg
3	4	621	393	1484	1096	133	00004.jpg
4	5	14	36	133	99	105	00005.jpg

Figure 7.20: Check the names in the test and train file column heads using the "head" function.

```
train_gen = CarsDataset(train_files,config.train_data,mode="train")
train_loader = DataLoader(train_gen,batch_size=config.batch_size,shuffle=True,pin_memory=True,num_workers=8)

val_gen = CarsDataset(test_files,config.test_data,augment=False,mode="train")
val_loader = DataLoader(val_gen,batch_size=config.batch_size,shuffle=False,pin_memory=True,num_workers=8)
```

Figure 7.21: Load images for train and test data.

7.6 TRANSFER LEARNING MODEL (EFFICIENTNET-B7)

EfficientNet is one of the few models that have state-of-the-art accuracy, and yet it is an order-of-magnitude smaller than the previous models [64]. The updated version of EfficientNet is based on the AutoML and Compounding Scaling. It is significantly faster compared to other models such as ResNet and Xception. Figure 7.22 shows the performance of EfficientNet-B7 on Imagenet in comparison to other models.

Now, we are going to load Efficiennet-B7 using PyTorch (Fig. 7.23). We are also going to keep track of the weights for backpropagation error calculation by setting the gradient requirement to True. Remember that, during the fine-tuning, we are going to set param.requires_grad = False (Check Section 7.5). Also, make sure to execute "`model.cuda( )`" to use GPU.

Now the pre-trained EfficietNet-B7 is loaded, and if you type "model" and execute it (by pressing Shift+Enter), you will be able to see the internal structure of the model. Here, we need to change the output from the fully connected layer. If you check at the bottom portion of the model, you will see that the output from the fully connected layer is 1,000 (Fig. 7.24).

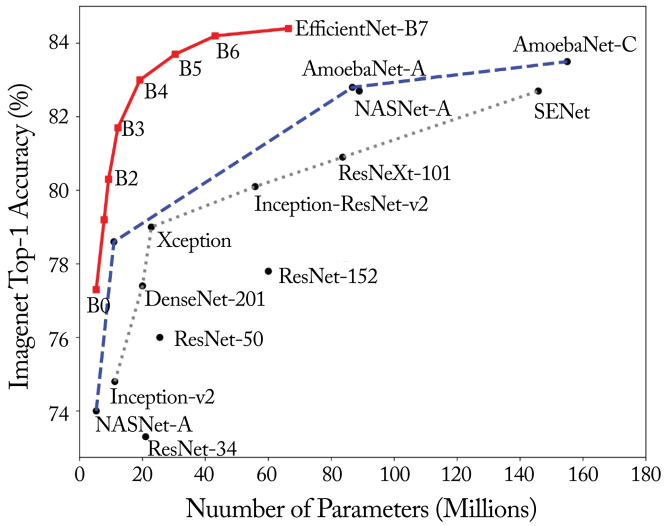

Figure 7.22: Model Size vs. ImageNet Accuracy. EfficientNet-B7 achieves new state-of-the-art 84.4% top-1 accuracy but being 8.4x smaller and 6.1x faster than GPipe. Also, EfficientNet-B1 is 7.6x smaller and 5.7x faster than ResNet-152 [64].

```
#import pre-trained EfficientNet
from efficientnet_pytorch import EfficientNet
model = EfficientNet.from_pretrained('efficientnet-b7')
```

```
Loaded pretrained weights for efficientnet-b7
```

```
for param in model.parameters():
    param.requires_grad = True
```

Figure 7.23: Load pre-trained network EfficientNet-B7. To save the gradients for backpropagation set param.requires_grad = True.

We will need to replace the output model classes, which is 196 different car models in this case study. Note that in Fig. 7.25, the fully connected layer output (out_features = config.num_classes) is defined. Now, if you check the model again, it will show that the fully connected layer has 196 output features. After that run `model.cuda( )` to use the default GPU.

Now, we are going to define the Stochastic Gradient Descent (SGD) using PyTorch "optim" function. Check the learning rate used in the DefaultConfig section. This learning rate is one of the hyperparameters that we could use to tune the model for our specific problem. A learning rate of 0.01 is a good starting point for many cases. As shown in Fig. 7.26, the loss

```
model
        3840, 640, kernel_size=(1, 1), stride=(1, 1), bias=False
        (static_padding): Identity()
      )
      (_bn2): BatchNorm2d(640, eps=0.001, momentum=0.010000000000000009, affine=True, track
_running_stats=True)
      (_swish): MemoryEfficientSwish()
    )
  )
  (_conv_head): Conv2dStaticSamePadding(
    640, 2560, kernel_size=(1, 1), stride=(1, 1), bias=False
    (static_padding): Identity()
  )
  (_bn1): BatchNorm2d(2560, eps=0.001, momentum=0.010000000000000009, affine=True, track_ru
nning_stats=True)
  (_avg_pooling): AdaptiveAvgPool2d(output_size=1)
  (_dropout): Dropout(p=0.5, inplace=False)
  (_fc): Linear(in_features=2560, out_features=1000, bias=True)
  (_swish): MemoryEfficientSwish()
)
```

Figure 7.24: Check the model architecture of EfficientNet-B7.

```
#for efficientnet-B7 only
model._fc = nn.Linear(in_features=2560, out_features=config.num_classes, bias=True)
```

```
model.cuda();
```

Figure 7.25: Change output features of the fully connected layer to 196.

```
optimizer = optim.SGD(model.parameters(),lr = config.lr,momentum=0.9,weight_decay=1e-4)
scheduler = lr_scheduler.StepLR(optimizer,step_size=10,gamma=0.1)
criterion = nn.CrossEntropyLoss().cuda()
```

Figure 7.26: Defining the optimizer, learning rate, and loss function.

function was assigned using "`torch.nn`" module. Here, we calculated the cross-entropy loss for training and testing images.

7.7 FINE-TUNING AND TRAINING

Finally, the model is ready for training. We will use a for loop here to run the model for a number of epochs. Here, we are going to run the model for 20 epochs.

An excellent way to run the model is to set the param.requries_grad = False (Fig. 7.23) and train the model for a low number of epochs, such as 2 (change epoch range inside for loop to 2). And then set the param.requires_grad = True for backpropagation (Fig. 7.23), run only that code segment lines (two lines in Fig. 7.23), and train the model for 20 epochs (Fig. 7.27). This small trick will help the model to learn better. Make sure that you are not loading the

```
print_every = 120
steps = 0
running_loss = 0
train_accuracy = 0
best_loss= np.inf
for epoch in range(20):
        model.train()
        scheduler.step(epoch)
        lr = get_learning_rate(optimizer)
        for images, labels in iter(train_loader):
            images, labels = images.cuda(), labels.cuda()
            steps += 1
            optimizer.zero_grad()
            output = model.forward(images)
            loss = criterion(output, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
            ps = torch.exp(output)
            equality = (labels.data == ps.max(dim=1)[1])
            train_accuracy += equality.type(torch.FloatTensor).mean()

            if steps % print_every == 0:
                model.eval()
                Test_loss = 0
                Test_accuracy = 0
                with torch.no_grad():
                    for images, labels in iter(val_loader):
                        images, labels = images.cuda(), labels.cuda()
                        output = model.forward(images)
                        Test_loss += criterion(output, labels).item()
                        ps = torch.exp(output)
                        equality = (labels.data == ps.max(dim=1)[1])
                        Test_accuracy += equality.type(torch.FloatTensor).mean()
                    is_best_loss = Test_loss < best_loss
                    best_loss= min(best_loss,Test_loss)
                    save_checkpoint({
                    "epoch":epoch + 1,
                    "state_dict":model.state_dict(),
                    "optimizer":optimizer.state_dict(),
                     },is_best_loss)
                print("Epoch: {}/{}.. ".format(epoch+1, config.epochs),
                      "Training Loss: {:.3f}.. ".format(running_loss/print_every),
                      "Training Accuracy: {:.3f}".format(train_accuracy/print_every),
                      "Testing Loss: {:.3f}.. ".format(Test_loss/len(val_loader)),
                      "Testing Accuracy: {:.3f}".format(Test_accuracy/len(val_loader)))

                running_loss = 0
                train_accuracy = 0
                model.train()
```

Figure 7.27: Code for training the model and print the accuracies.

EfficientNet again. If you do that, you will lose the saved weights for the first 2 epochs. You may also update the epoch numbers in the code in Fig. 7.9, but it will affect only the output display during training.

When you run the model for 2 epochs, you will see that the training and testing accuracy are continuously improving. After 2 epochs, the testing accuracy is 19.5% (Fig. 7.28).

For the next run, when 20 epochs are used, testing accuracy starts from 28%. Because at this time, it is using the updated best parameters from 2 epochs. Make sure to set the gradient requirement to "True" and run that portion only (Fig. 7.23) before the second run.

Finally, after 20 epochs, we will get an accuracy higher than 90%. Here, the model will save only the parameters that give higher accuracy, which is 90.6% for this simulation.

```
Epoch: 1/2..  Training Loss: 5.257..  Training Accuracy: 0.015 Testing Loss: 5.185..  Testing Accuracy: 0.041
Epoch: 1/2..  Training Loss: 5.174..  Training Accuracy: 0.029 Testing Loss: 5.044..  Testing Accuracy: 0.083
Epoch: 1/2..  Training Loss: 5.098..  Training Accuracy: 0.044 Testing Loss: 4.937..  Testing Accuracy: 0.108
Epoch: 1/2..  Training Loss: 4.994..  Training Accuracy: 0.056 Testing Loss: 4.832..  Testing Accuracy: 0.141
Epoch: 2/2..  Training Loss: 4.823..  Training Accuracy: 0.107 Testing Loss: 4.739..  Testing Accuracy: 0.165
Epoch: 2/2..  Training Loss: 4.740..  Training Accuracy: 0.123 Testing Loss: 4.657..  Testing Accuracy: 0.174
Epoch: 2/2..  Training Loss: 4.696..  Training Accuracy: 0.117 Testing Loss: 4.578..  Testing Accuracy: 0.183
Epoch: 2/2..  Training Loss: 4.588..  Training Accuracy: 0.132 Testing Loss: 4.491..  Testing Accuracy: 0.195
```

After running the model for 2 epochs, change the epochs to 20 inside the for loop, "for epoch in range(2)→for epoch in range(20)", set param.requires_grad = True (execute command) and run the model again.

```
Epoch: 1/20..  Training Loss: 3.883..  Training Accuracy: 0.232 Testing Loss: 3.200..  Testing Accuracy: 0.280
Epoch: 1/20..  Training Loss: 3.039..  Training Accuracy: 0.317 Testing Loss: 2.362..  Testing Accuracy: 0.440
Epoch: 1/20..  Training Loss: 2.388..  Training Accuracy: 0.415 Testing Loss: 1.939..  Testing Accuracy: 0.525
Epoch: 1/20..  Training Loss: 1.915..  Training Accuracy: 0.515 Testing Loss: 1.421..  Testing Accuracy: 0.629
Epoch: 2/20..  Training Loss: 1.388..  Training Accuracy: 0.648 Testing Loss: 1.196..  Testing Accuracy: 0.680
Epoch: 2/20..  Training Loss: 1.148..  Training Accuracy: 0.693 Testing Loss: 1.040..  Testing Accuracy: 0.719
Epoch: 2/20..  Training Loss: 1.077..  Training Accuracy: 0.713 Testing Loss: 0.975..  Testing Accuracy: 0.735
Epoch: 2/20..  Training Loss: 1.004..  Training Accuracy: 0.726 Testing Loss: 0.838..  Testing Accuracy: 0.764
Epoch: 3/20..  Training Loss: 0.746..  Training Accuracy: 0.795 Testing Loss: 0.793..  Testing Accuracy: 0.781
Epoch: 3/20..  Training Loss: 0.633..  Training Accuracy: 0.823 Testing Loss: 0.756..  Testing Accuracy: 0.783
Epoch: 3/20..  Training Loss: 0.626..  Training Accuracy: 0.823 Testing Loss: 0.685..  Testing Accuracy: 0.806
Epoch: 3/20..  Training Loss: 0.586..  Training Accuracy: 0.842 Testing Loss: 0.631..  Testing Accuracy: 0.818
Epoch: 4/20..  Training Loss: 0.554..  Training Accuracy: 0.843 Testing Loss: 0.604..  Testing Accuracy: 0.826
Epoch: 4/20..  Training Loss: 0.386..  Training Accuracy: 0.896 Testing Loss: 0.569..  Testing Accuracy: 0.841
Epoch: 4/20..  Training Loss: 0.413..  Training Accuracy: 0.880 Testing Loss: 0.578..  Testing Accuracy: 0.839
Epoch: 4/20..  Training Loss: 0.386..  Training Accuracy: 0.894 Testing Loss: 0.565..  Testing Accuracy: 0.837
```

Figure 7.28: Output display for training and testing losses and accuracies.

```
Epoch: 18/20..  Training Loss: 0.035..  Training Accuracy: 0.994 Testing Loss: 0.374..  Testing Accuracy: 0.903
Epoch: 18/20..  Training Loss: 0.034..  Training Accuracy: 0.994 Testing Loss: 0.374..  Testing Accuracy: 0.905
Epoch: 19/20..  Training Loss: 0.032..  Training Accuracy: 0.994 Testing Loss: 0.373..  Testing Accuracy: 0.905
Epoch: 19/20..  Training Loss: 0.036..  Training Accuracy: 0.991 Testing Loss: 0.371..  Testing Accuracy: 0.905
Epoch: 19/20..  Training Loss: 0.027..  Training Accuracy: 0.993 Testing Loss: 0.370..  Testing Accuracy: 0.904
Epoch: 19/20..  Training Loss: 0.033..  Training Accuracy: 0.993 Testing Loss: 0.371..  Testing Accuracy: 0.904
Epoch: 20/20..  Training Loss: 0.028..  Training Accuracy: 0.994 Testing Loss: 0.368..  Testing Accuracy: 0.906
Epoch: 20/20..  Training Loss: 0.034..  Training Accuracy: 0.992 Testing Loss: 0.367..  Testing Accuracy: 0.905
```

Figure 7.29: Output display for training and testing losses and accuracies after 20 epochs (Learning rate 0.01%).

If you have a situation where training accuracy is increasing, but the testing accuracy is decreasing, you will have to stop the training because that is a sign of overfitting training data.

Now, to improve the model performance, we can further change some of the training parameters that affect the learning process, such as learning rate, epochs, number of hidden layers, etc. This is called hyperparameters tuning, and this step should be done carefully to see how the model is learning. For example, if we change the learning rate from 0.01 to 0.025 (Fig. 7.9) and run the whole simulation as described in this section, we will get a better result (testing accuracy 91.5%). The simulation output for learning rate 0.025 is shown in Fig. 7.30. We may also check different learning rates and hyperparameters to improve testing accuracy.

At this point, we can say that our deep learning model will be able to predict the make, model, and year of a car within 91.5% accuracy.

```
Epoch: 1/2..  Training Loss: 5.231..  Training Accuracy: 0.020 Testing Loss: 5.058..  Testing Accuracy: 0.060
Epoch: 1/2..  Training Loss: 5.061..  Training Accuracy: 0.043 Testing Loss: 4.766..  Testing Accuracy: 0.109
Epoch: 1/2..  Training Loss: 4.923..  Training Accuracy: 0.055 Testing Loss: 4.569..  Testing Accuracy: 0.132
Epoch: 1/2..  Training Loss: 4.726..  Training Accuracy: 0.080 Testing Loss: 4.384..  Testing Accuracy: 0.163
Epoch: 2/2..  Training Loss: 4.398..  Training Accuracy: 0.146 Testing Loss: 4.243..  Testing Accuracy: 0.184
Epoch: 2/2..  Training Loss: 4.290..  Training Accuracy: 0.141 Testing Loss: 4.132..  Testing Accuracy: 0.199
Epoch: 2/2..  Training Loss: 4.261..  Training Accuracy: 0.139 Testing Loss: 4.025..  Testing Accuracy: 0.210
Epoch: 2/2..  Training Loss: 4.129..  Training Accuracy: 0.156 Testing Loss: 3.908..  Testing Accuracy: 0.219
```

Learning rate changed to 0.025.
After running the model for 2 epochs, change the epochs to 20 inside the for loop, "for epoch in range(2)→for epoch in range(20)", set param.requires_grad = True (execute command) and run the model again.

```
Epoch: 1/2..  Training Loss: 5.231..  Training Accuracy: 0.020 Testing Loss: 5.058..  Testing Accuracy: 0.060
Epoch: 1/2..  Training Loss: 5.061..  Training Accuracy: 0.043 Testing Loss: 4.766..  Testing Accuracy: 0.109
Epoch: 1/2..  Training Loss: 4.923..  Training Accuracy: 0.055 Testing Loss: 4.569..  Testing Accuracy: 0.132
Epoch: 1/2..  Training Loss: 4.726..  Training Accuracy: 0.080 Testing Loss: 4.384..  Testing Accuracy: 0.163
Epoch: 2/2..  Training Loss: 4.398..  Training Accuracy: 0.146 Testing Loss: 4.243..  Testing Accuracy: 0.184
Epoch: 2/2..  Training Loss: 4.290..  Training Accuracy: 0.141 Testing Loss: 4.132..  Testing Accuracy: 0.199
Epoch: 2/2..  Training Loss: 4.261..  Training Accuracy: 0.139 Testing Loss: 4.025..  Testing Accuracy: 0.210
Epoch: 2/2..  Training Loss: 4.129..  Training Accuracy: 0.156 Testing Loss: 3.908..  Testing Accuracy: 0.219
⋮
Epoch: 18/20..  Training Loss: 0.027..  Training Accuracy: 0.994 Testing Loss: 0.351..  Testing Accuracy: 0.912
Epoch: 18/20..  Training Loss: 0.029..  Training Accuracy: 0.993 Testing Loss: 0.349..  Testing Accuracy: 0.913
Epoch: 19/20..  Training Loss: 0.032..  Training Accuracy: 0.990 Testing Loss: 0.349..  Testing Accuracy: 0.913
Epoch: 19/20..  Training Loss: 0.036..  Training Accuracy: 0.993 Testing Loss: 0.347..  Testing Accuracy: 0.913
Epoch: 19/20..  Training Loss: 0.024..  Training Accuracy: 0.993 Testing Loss: 0.347..  Testing Accuracy: 0.913
Epoch: 19/20..  Training Loss: 0.030..  Training Accuracy: 0.992 Testing Loss: 0.349..  Testing Accuracy: 0.913
Epoch: 20/20..  Training Loss: 0.028..  Training Accuracy: 0.993 Testing Loss: 0.348..  Testing Accuracy: 0.914
Epoch: 20/20..  Training Loss: 0.031..  Training Accuracy: 0.992 Testing Loss: 0.347..  Testing Accuracy: 0.913
Epoch: 20/20..  Training Loss: 0.028..  Training Accuracy: 0.992 Testing Loss: 0.348..  Testing Accuracy: 0.915
Epoch: 20/20..  Training Loss: 0.024..  Training Accuracy: 0.994 Testing Loss: 0.350..  Testing Accuracy: 0.913
```

Figure 7.30: Output display for training and testing losses and accuracies after 20 epochs (Learning rate 0.025%).

7.8 MODEL TESTING

The testing accuracy defines how well the model can predict the testing data set, which has 8,041 car images. After training, our model can predict 91.5% of images correctly. However, this accuracy is based on the testing images provided by Stanford's car dataset. We may check the model performance by taking a car picture or by using a random car image from the internet.

We did a google search "Volkswagen Golf Hatchback 1991," saved the image, and uploaded it to the GCP cloud home folder using Jupyter Notebook. We also did cropping before classifying this image, since our model was trained on cropped images. Figure 7.31 shows the way to read an image from the home folder (e.g., car_image1.jpg) and resize it for testing using our model.

Now to test this image through the model, we used the codes shown in Fig. 7.32. The model output class here is 190+1 =191. As Python reads columns from 0, we should add 1. From the cars_meta.csv file, we also find that class 191 is actually Volkswagen Golf Hatchback 1991.

We also checked a couple of pre-2013 car model images from the internet, and most of the time (9 out of 10), the model can predict car make, model, and year correctly.

```
image = cv2.imread('car_image1.jpg',1)
image  = cv2.resize (image,(224,224))
```

```
import matplotlib.pyplot as plt
plt.imshow(image)
```

```
<matplotlib.image.AxesImage at 0x7f397bae77f0>
```

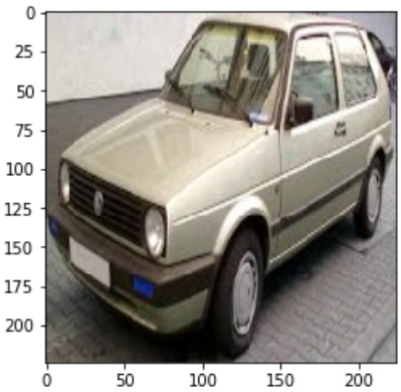

Figure 7.31: Codes to read car image and resize.

```
image = T.Compose([T.ToPILImage(),T.ToTensor(),T.Normalize([0.485, 0.456, 0.406],[0.229, 0.224, 0.225])])(image)
image = image.unsqueeze(0)
image = image.float()
```

```
image = image.cuda(non_blocking=True)
model.eval();
output = model(image)
ps = torch.exp(output)
ps.max(dim=1)[1]
```

```
tensor([190], device='cuda:0')
```

Figure 7.32: Load car image and test the model output. The output class 191 (190+1) is for Volkswagen Golf Hatchback 1991 model.

If you have followed all the steps up to here, you should also try to check this model performance on different car images. You may also try testing images randomly from 8,041 given images (test dataset) to see if the model can predict the car make, types, and year correctly or not.

7.9 CONCLUSION

In this short book, we have shown a practical case study on how to use Python and deep learning models for classifying image datasets. The accuracy we have achieved here for multiclass clas-

sification is relatively good. Typically, if you spend a significant amount of time on optimizing and do a lot of trials with hyperparameters, you will get an accuracy higher than 90%. But it also depends on the type and quality of the dataset you use for training. A good quality data for deep learning should be large in volume, organized, and cleaned.

The Python programming used in the case study problem is a little advanced-level. You might find a gap in the difficulty level between the examples showed in the first two chapters and the case study. If you come across any Python library and program syntax that looks unfamiliar to you, you should review or study that from another source. There are a lot of free online Python books and useful learning resources available that you can use for this purpose.

The goal of this book was to give someone a jump-start in the journey of deep learning, without covering too many details and theories. One of the significant challenges in implementing a deep learning algorithm is that it requires a GPU enabled compute engine. To overcome this obstacle, we presented a cost-effective approach (preemptible) for using GPU in a cloud platform (GCP).

The program shown here is structured in such a way that it can be used for other deep learning models from PyTorch. The best way to use the instructions in this book is to utilize free datasets to train a deep learning model that is available online. Finally, if you are doing any computer-vision related project, you may take a lot of pictures by yourself and use your dataset for training.

Deep learning or artificial intelligence is going to make a significant impact on every aspect of our life. If we could blend our engineering knowledge with this new technology, it will surely assist us in doing many exciting projects and solving prediction related problems.

Bibliography

[1] G. Piatetsky, Python overtakes R, becomes the leader in data science, machine learning platform, 2017. https://www.kdnuggets.com/2017/08/python-overtakes-r-leader-analytics-data-science.html 1

[2] P. Guo, Python is now the most popular introductory teaching language at top U.S. Universities, https://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-u-s-universities/fulltext 1

[3] J. M. Perkel, Why Jupyter is data scientists' computational notebook of choice, *Nature*, 563(7729):145–146, November 2018. DOI: 10.1038/d41586-018-07196-1. 2

[4] PyTorch documentation. https://pytorch.org/docs/stable/index.html 11

[5] F. Chollet, *Deep Learning with Python*, Manning Publications Company, 2017. 11, 24

[6] Tensor processing unit. https://en.wikipedia.org/wiki/Tensor_processing_unit 11

[7] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, and H. Arshad, State-of-the-art in artificial neural network applications: A survey, *Heliyon*, 4(11):e00938, 2018. DOI: 10.1016/j.heliyon.2018.e00938. 14, 18

[8] M. Mehdy, P. Ng, E. Shair, N. Saleh, and C. Gomes, Artificial neural networks in image processing for early detection of breast cancer, *Computational and Mathematical Methods in Medicine*, 2017. DOI: 10.1155/2017/2610628. 18

[9] O. Araque, I. Corcuera-Platas, J. F. Sánchez-Rada, and C. A. Iglesias, Enhancing deep learning sentiment analysis with ensemble techniques in social applications, *Expert Systems with Applications*, 77:236–246, 2017. DOI: 10.1016/j.eswa.2017.02.002. 18

[10] X. Zhou, W. Gong, W. Fu, and F. Du, Application of deep learning in object detection, *IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS)*, pages 631–634, 2017. DOI: 10.1109/icis.2017.7960069. 18

[11] Activation function. https://en.wikipedia.org/wiki/Activation_function 24

[12] Y. Ho and S. Wookey, The real-world-weight cross-entropy loss function: Modeling the costs of mislabeling, *IEEE Access*, 2019. DOI: 10.1109/access.2019.2962617. 24

[13] C. C. Aggarwal, *Neural Networks and Deep Learning: A Textbook*, Springer International Publishing, 2018. DOI: 10.1007/978-3-319-94463-0. 25, 36

[14] A. Ng, CS229 lecture notes: Supervised learning, 2018. http://cs229.stanford.edu/notes/cs229-notes1.pdf 26

[15] E. Kayacan and M. A. Khanesar, Chapter 5—Gradient descent methods for type-2 fuzzy neural networks, *Fuzzy Neural Networks for Real Time Control Applications*, pages 45–70, E. Kayacan and M. A. Khanesar, Eds., Butterworth-Heinemann, 2016. DOI: 10.1016/c2014-0-02444-6. 27

[16] S. Lau, Learning rate schedules and adaptive learning rate methods for deep learning, 2017. https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1 27

[17] PyTorch documentation—TORCH.OPTIM. https://pytorch.org/docs/stable/optim.html 27

[18] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, Rethinking the inception architecture for computer vision, *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016. DOI: 10.1109/cvpr.2016.308. 29

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton, ImageNet classification with deep convolutional neural networks, *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012. DOI: 10.1145/3065386. 29

[20] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, Inception-v4, Inception-ResNet and the impact of residual connections on learning, *31st AAAI Conference on Artificial Intelligence*, 2017. 29

[21] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, Densely connected convolutional networks, *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4700–4708, 2017. DOI: 10.1109/cvpr.2017.243. 29

[22] D. Han, J. Kim, and J. Kim, *Deep Pyramidal Residual Networks*, pages 6307–6315, 2017. DOI: 10.1109/cvpr.2017.668. 29

[23] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016. DOI: 10.1145/2976749.2978318. 29

[24] Y. LeCun, Y. Bengio, and G. Hinton, Deep learning, *Nature*, 521(7553):436–444, May 1, 2015. DOI: 10.1038/nature14539. 29

[25] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, Gradient-based learning applied to document recognition, *Proc. of the IEEE*, 86(11):2278–2324, 1998. DOI: 10.1109/5.726791. 30

[26] Y. Kim, Convolutional neural networks for sentence classification, *Proc. of the Conference on Empirical Methods in Natural Language Processing*, August 25, 2014. DOI: 10.3115/v1/d14-1181. 30

[27] O. Abdel-Hamid, A. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, Convolutional neural networks for speech recognition, *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 22(10):1533–1545, 2014. DOI: 10.1109/taslp.2014.2339736. 30

[28] D. Ciregan, U. Meier, and J. Schmidhuber, Multi-column deep neural networks for image classification, *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3642–3649, 2012. DOI: 10.1109/cvpr.2012.6248110. 30

[29] X. Liu, Z. Deng, and Y. Yang, Recent progress in semantic image segmentation, *Artificial Intelligence Review*, 52(2):1089–1106, August 1, 2019. DOI: 10.1007/s10462-018-9641-3. 30

[30] J. Kim, V. D. Calhoun, E. Shim, and J. H. Lee, Deep neural network with weight sparsity control and pre-training extracts hierarchical features and enhances classification performance: Evidence from whole-brain resting-state functional connectivity patterns of schizophrenia, *Neuroimage*, 124(PtA):127–146, January 1, 2016. DOI: 10.1016/j.neuroimage.2015.05.018. 32

[31] Y. Zhao et al., Automatic recognition of fMRI-derived functional networks using 3-D convolutional neural networks, *IEEE Transactions on Biomedical Engineering* 65(9):1975–1984, September 2018. DOI: 10.1109/tbme.2017.2715281. 32

[32] H. Jang, S. M. Plis, V. D. Calhoun, and J. H. Lee, Task-specific feature extraction and classification of fMRI volumes using a deep neural network initialized with a deep belief network: Evaluation using sensorimotor tasks, *Neuroimage*, 145(PtB):314–328, January 15, 2017. DOI: 10.1016/j.neuroimage.2016.04.003. 32

[33] X. Li and X. Wu, Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition, *IEEE.* October 15, 2014. DOI: 10.1109/icassp.2015.7178826. 33

[34] A. Graves, A. Mohamed, and G. Hinton, Speech recognition with deep recurrent neural networks, *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649, 2013. DOI: 10.1109/icassp.2013.6638947. 33, 34

[35] D. Güera and E. J. Delp, Deepfake video detection using recurrent neural networks, *15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pages 1–6, 2018. DOI: 10.1109/avss.2018.8639163. 33

[36] A.-I. Marinescu, Bach 2.0—generating classical music using recurrent neural networks, *Proc. Computer Science*, 159:117–124, January 1, 2019. DOI: 10.1016/j.procs.2019.09.166. 33

[37] D. Eck and J. Schmidhuber, Learning the long-term structure of the blues, *Artificial Neural Networks, (ICANN)*, pages 284–289, Berlin, Heidelberg, Springer Berlin Heidelberg, 2002. DOI: 10.1007/3-540-46084-5_47. 33

[38] S. Lawrence, C. L. Giles, and S. Fong, Natural language grammatical inference with recurrent neural networks, *Knowledge and Data Engineering, IEEE Transactions on*, 12:126–140, February 1, 2000. DOI: 10.1109/69.842255. 33

[39] M.-J. Zhang and Z.-Z. Chu, Adaptive sliding mode control based on local recurrent neural networks for underwater robot, *Ocean Engineering*, 45:56–62, May 1, 2012. DOI: 10.1016/j.oceaneng.2012.02.004. 33

[40] A. Chinea Manrique de Lara, Understanding the principles of recursive neural networks: A generative approach to tackle model complexity, *Artificial Neural Networks–ICANN 2009*. DOI: 10.1007/978-3-642-04274-4_98. 35

[41] X. Wang and A. Gupta, Unsupervised learning of visual representations using videos, *IEEE International Conference on Computer Vision (ICCV)*, pages 2794–2802, 2015. DOI: 10.1109/iccv.2015.320. 35

[42] A. Sagheer and M. Kotb, Unsupervised pre-training of a deep LSTM-based stacked autoencoder for multivariate time series forecasting problems, *Scientific Reports*, 9(1):19038, December 13, 2019. DOI: 10.1038/s41598-019-55320-6. 36

[43] J. Hui, GAN—-why it is so hard to train generative adversarial networks!, 2018. https://medium.com/@jonathan_hui/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b 36

[44] Y. Hong, Comparison of generative adversarial networks architectures which reduce mode collapse, 2019. *arXiv:1910.04636* 36

[45] K. Schawinski, C. Zhang, H. Zhang, L. Fowler, and G. K. Santhanam, Generative adversarial networks recover features in astrophysical images of galaxies beyond the deconvolution limit, *Monthly Notices of the Royal Astronomical Society: Letters*, 467(1):L110–L114, 2017. DOI: 10.1093/mnrasl/slx008. 36

[46] H. C. Shin et al., Deep convolutional neural networks for computer-aided detection: CNN architectures, dataset characteristics and transfer learning, *IEEE Transactions on Medical Imaging*, 35(5):1285–98, May 2016. DOI: 10.1109/tmi.2016.2528162. 37

[47] R. Marée, P. Geurts, and L. Wehenkel, Towards generic image classification using tree-based learning: An extensive empirical study, *Pattern Recognition Letters*, 74:17–23, April 15, 2016. DOI: 10.1016/j.patrec.2016.01.006. 37

[48] O. Russakovsky et al., ImageNet large scale visual recognition challenge, *International Journal of Computer Vision*, 115(3):211–252, December 1, 2015. DOI: 10.1007/s11263-015-0816-y. 37

[49] A. Torralba, R. Fergus, and W. T. Freeman, 80 million tiny images: A large data set for nonparametric object and scene recognition, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(11):1958–1970, 2008. DOI: 10.1109/tpami.2008.128. 37

[50] S. J. Pan and Q. Yang, A survey on transfer learning, *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010. DOI: 10.1109/tkde.2009.191. 37

[51] Keras applications. https://keras.io/api/applications/ 38

[52] Q. V. Le, Building high-level features using large scale unsupervised learning, *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8595–8598, 2013. DOI: 10.1109/icassp.2013.6639343. 38

[53] A. Radford, L. Metz, and S. Chintala, Unsupervised representation learning with deep convolutional generative adversarial networks, *ArXiv Preprint ArXiv:1511.06434*, 2015. 38

[54] R. Wan, H. Xiong, X. Li, Z. Zhu, and J. Huan, Towards making deep transfer learning never hurt, *IEEE International Conference on Data Mining (ICDM)*, pages 578–587, 2019. DOI: 10.1109/icdm.2019.00068. 38

[55] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, A survey on deep transfer learning, *International Conference on Artificial Neural Networks*, pages 270–279, Springer, 2018. DOI: 10.1007/978-3-030-01424-7_27. 38

[56] TORCHVISION.MODELS. https://pytorch.org/docs/stable/torchvision/models.html 39

[57] Google app engine documentation. https://cloud.google.com/appengine/docs/ 41

[58] Google cloud free tier. https://cloud.google.com/free/docs/gcp-free-tier 41

[59] Virtual machine instances. https://cloud.google.com/compute/docs/instances 45

[60] Preemptible VM instances. https://cloud.google.com/compute/docs/instances/preemptible 54

[61] Running a notebook server. https://jupyter-notebook.readthedocs.io/en/stable/public_server.html 62

[62] Cars dataset. http://ai.stanford.edu/jkrause/cars/car_dataset.html 67

[63] J. Krause, M. Stark, J. Deng, and L. Fei-Fei, 3-D object representations for fine-grained categorization, *IEEE International Conference on Computer Vision Workshops*, pages 554–561, 2013. DOI: 10.1109/iccvw.2013.77. 67

[64] M. Tan and Q. V. Le, EfficientNet: Rethinking model scaling for convolutional neural networks, *ArXiv Preprint ArXiv:1905.11946*, 2019. 79, 80

Author's Biography

TARIQ M. ARIF

Tariq M. Arif is an assistant professor in the Department of Mechanical Engineering at Weber State University, UT. Prior to that, he worked at the University of Wisconsin, Platteville, as a lecturer. Tariq obtained his Ph.D. in 2017 from the Mechanical Engineering department of the New Jersey Institute of Technology (NJIT), NJ. His main research interests are in the area of artificial intelligence and genetic algorithm for robotics control, computer vision, and biomedical simulations of focused ultrasound surgery. He completed his Masters in 2011 from the University of Tokushima, Japan, and a B.Sc. in 2005 from Bangladesh University of Engineering and Technology (BUET). Tariq also worked in the Japanese automobile industry as a CAD/CAE engineer after completing his B.Sc. degree. In his industrial and academic carrier, Tariq has been involved in many different research projects. Currently, he is working on the implementation of deep learning models for various engineering tasks.