AI PUBLISHING

Deep Learning with Python

Practical Guide for Beginners

Learning AI and
Data Science easily

Our Books are designed
to teach beginners
Data Science and AI

DEEP LEARNING
WITH PYTHON
FOR BEGINNERS

AI PUBLISHING

First Printing, 2019

Edited by AI Publishing
Ebook Converted and Cover by Gazler Studio
Published by AI Publishing LLC

ISBN-13: 978-1-7330426-5-9
ISBN-10: 1-7330426-5-9

How to contact us

If you have any feedback, please let us know by sending an email to contact@aispublishing.net.

This feedback is highly valued, and we look forward to hearing from you. It will be very helpful for us to improve the quality of our books.

To get the Python codes and materials used in this book, please click the link below:

https://www.aispublishing.net/mastering-deep-learning-with-python

About the Publisher

At AI Publishing Company, we have established an international learning platform specifically for young students, beginners, small enterprises, startups, and managers who are new to data sciences and artificial intelligence.

Through our interactive, coherent, and practical books and courses, we help beginners learn skills that are crucial to developing AI and data science projects.

Our courses and books range from basic intro courses to language programming and data sciences to advanced courses for machine learning, deep learning, computer vision, big data, and much more, using programming languages like Python, R, and some data science and AI software.

AI Publishing's core focus is to enable our learners to create and try proactive solutions for digital problems by leveraging the power of AI and data sciences to the maximum extent.

Moreover, we offer specialized assistance in the form of our free online content and ebooks, providing up-to-date and useful insight into AI practices and data-science subjects, along with eliminating the doubts and misconceptions about AI and programming.

Our experts have cautiously developed our online courses and kept them concise, short, and comprehensive so that you can understand everything clearly and effectively and start practicing the applications right away.

We also offer consultancy and corporate training in AI and data sciences for enterprises, so that their staff can navigate through the workflow efficiently.

With AI Publishing, you can always stay closer to the innovative world of AI and data sciences.

If you are also eager to learn the A to Z of AI and data sciences but have no clue where to start, AI Publishing is the finest place to go.

Please contact us by email at: contact@aispublishing.net.

AI Publishing is searching for author like you

If you're interested in becoming an author for AI Publishing, please contact us at authors@aispublishing.net.

We are working with developers and AI tech professionals, just like you, to help them share their insight with the global AI and Data Science lovers. You can share all subject about hot topic in AI and Data Science.

Book Approach

The book provides hands-on examples to grow an intuitive understanding of deep learning. While you can read this book without picking up your laptop, we highly recommend you experiment with the practical part available online as Jupyter notebooks at:

https://www.aispublishing.net/mastering-deep-learning-with-python

Download the color images

We also provide a PDF file that has color images of the screenshots/diagrams used in this book. You can download it here:

https://www.aispublishing.net/book-dl-python

Get in touch with us

Feedback from our readers is always welcome.

For general feedback please send us an Email at
contact@aipublishing.net
and mention the book title in the subject of your message.

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you could report this to us as soon as you can.

If you are interested in becoming an AI Publishing author: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please send us a Email at
author@aipublishing.net

Table of Contents

Preface

§ Who should read this book?

This book is written for beginners and novices who want to develop fundamental data science skills and learn how to build models that learn useful information from data. This book will prepare the learner for a career or further learning that involves more advanced topics. It contains an introduction and fundamental concepts used in data science and deep learning. The learner does not need to have any prior knowledge of machine learning or deep learning, but some basic understanding of mathematics is required.

§ Why this book?

This book focuses on the practical implementation of deep learning algorithms.

The Jupyter notebooks for each topic is available in the link below. So, you can execute the code and understand the working of the algorithm step by step.

https://www.aispublishing.net/book-dl-python

Each chapter begins with an explanation of the chapter's content relevant to data science.

§ What are data science and deep learning?

Today, we are bombarded with information that is generated through machines in all corners of the world. From surveillance cameras, GPS trackers, satellites, and search engines to our mobile phones and smart appliances in the kitchen, all these entities generate some kind of data. Usually, it contains information about users: their routines, their likes and dislikes, their choices, or even their work hours.

The most important reason for the growth of machine learning in recent years is the exponential growth of available data and computing power. Surveillance cameras, GPS trackers, satellites, social media, and millions of other such entities generate data. Data about users' habits, routines, likes, and dislikes is collected through various apps and during web surfing.

So out of all this data, we need to extract useful and relevant information, and this is what data science is all about. Data science is actually *making sense of the data*.

Today, the research is more focused on making sense of this data and extracting useful information from it. By collecting and analyzing large-scale data, not only can we develop useful applications, but we can also tailor the application for personalized use as per each user's needs. Statistics and probability provide the basis to carry out data analysis in data science. These play a crucial role and are the most important requirements to learn about.

§ Data science applications

Data science has been applied to a vast range of domains, like finance, education, business, and healthcare. Data science is a powerful tool in fighting cancer, diabetes, and various heart diseases. Machine learning algorithms are being employed to recognize specific patterns for symptoms of these conditions. Some machine learning models can even predict the chance of having a heart attack within a particular time frame. Cancer researchers are using deep learning models to detect cancer cells. Research is being conducted at UCLA to identify cancer cells using deep learning.

Deep learning models have been built that accurately detect and recognize faces in real time. Through such models, social media applications like Facebook and Twitter can quickly recognize the faces in uploaded images and can automatically tag them. Such applications are also being used for security purposes.

Speech recognition is another major success and a dynamic area of research. The machine learns to recognize the voice of a person, can also convert the spoken words to text, and can understand the meaning of those words to get the command.

One of the hottest research areas is self-driving cars. The car learns to drive as it interacts with its environment, using data from the camera and various sensors. Those cars use deep learning and learn to recognize and understand a stop sign, differentiate between a pedestrian and a lampost, and avoid collision with other vehicles.

1

Python Basics

The subsequent chapters contain an introduction to the basic concepts. This includes the basics of Python, NumPy, Pandas, and OpenCV. We also give an introduction to plotting using Matplotlib as it is used for various visualizations, including but not limited to displaying images and plotting results.

1.1. Python

Data containers are different data structures that we use to store data in various forms. Here, we will learn about data containers in Python. It is necessary that we understand their use because the data is stored in those containers, and we should know how to use them properly and when to use them.

1.2. Data Containers

Here, we will discuss all three data containers: Lists, Tuples, and Dictionaries.

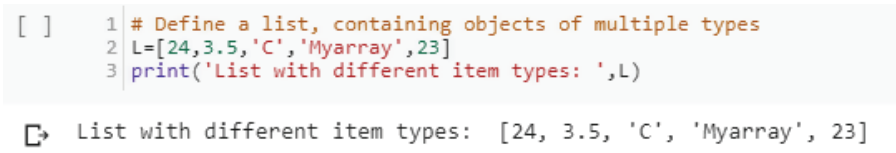
1.2.1. Lists

List, as the name suggests, is a list of values. It contains items enclosed within square brackets. The difference between a

regular array (in C++) and a list (in Python) is that the items in a list can be of different types. For example, the following is a list.

PYTHON CODE:

```
# Define a list, containing objects of multiple types
L=[24,3.5,'C','Myarray',23]
print('List with different item types: ',L)
```



The elements in the list can be accessed through their positions (also called indexes). Indexing of elements in a list starts with 0 and goes to length −1. The slice operator [:] is used to select a sublist from a bigger list. In a slice operator, the first index shows an element of the starting index. The range is open-ended from the end side, i.e., the last value in a slice operator is excluded from the range. Some built-in list functions are:

- **append(val)**: Appends the value "val" at the end of the list
- **insert(index,val)**: Adds a new values "val" at index
- **remove(val)**: Removes the first occurrence of the value from the list
- **pop()**: Removes and returns the last element from the list
- **sort()**: Sorts the list

PYTHON CODE:

```
print('Complete list L : ',L)
print('Element at index 3: ',L[3])
print('All elements from index 1 onwards: ',L[1:])
print('All elements before index 3(excluding index 3):
',L[:3])
print('Elements from index 2 to index 4: ',L[2:5])

# To repeat an element multiple times, we use * operator a
number
# after * defines the number of times to repeat the list

print('Reptition of [4] 3 times: ',[4]*3)

# Now, we will modify the list elements:

print('\nUpdate list L: change the value of 3.5 to 6.5:')

# Assigning second element of the list a value of 6.5

L[1]=6.5

print('Complete list: ',L)

L.append(45)
print('\nAdding new element (45) at the end of L: ',L)

L.insert(3,'myname')
print('\nInserted a new element at index 3:',L)

L.remove('C')
print('\nAfter removing character \'C\' from the list:',L )

L.pop()
print('\nAfter popping last element from list:',L)
```

```
Complete list L :  [24, 3.5, 'C', 'Myarray', 23]
Element at index 3:  Myarray
All elements from index 1 onwards:  [3.5, 'C', 'Myarray', 23]
All elements before index 3(excluding index 3):  [24, 3.5, 'C']
Elements from index 2 to index 4:  ['C', 'Myarray', 23]
Reptition of [4] 3 times:  [4, 4, 4]

Update list L: change the value of 3.5 to 6.5:
Complete list:  [24, 6.5, 'C', 'Myarray', 23]

Adding new element (45) at the end of L:  [24, 6.5, 'C', 'Myarray', 23, 45]

Inserted a new element at index 3: [24, 6.5, 'C', 'myname', 'Myarray', 23, 45]

After removing character 'C' from the list: [24, 6.5, 'myname', 'Myarray', 23, 45]

After popping last element from list: [24, 6.5, 'myname', 'Myarray', 23]
```

1.2.2. Arithmetic Operators on Lists

Following are the two operations performed on the lists:

- + will combine two lists
- will repeat a list

PYTHON CODE:

```
# Define first list
list1=[1,5,10,15]

# Define second list
list2=[2,4,8,12]

print('List 1:',list1)
print('List 2:',list2)

# Combine two lists
print('\nCombining two lists:',list1+list2)

# Repeat a list 3 times
print('\nRepeat list1 3 times:',list1*3)
```

```
[ ]
# Define first list
list1=[1,5,10,15]

# Define second list
list2=[2,4,8,12]

print('List 1:',list1)
print('List 2:',list2)

# Combine two lists
print('\nCombining two lists:',list1+list2)

# Repeat a list 3 times
print('\nRepeat list1 3 times:',list1*3)
```

```
List 1: [1, 5, 10, 15]
List 2: [2, 4, 8, 12]

Combining two lists: [1, 5, 10, 15, 2, 4, 8, 12]

Repeat list1 3 times: [1, 5, 10, 15, 1, 5, 10, 15, 1, 5, 10, 15]
```

1.2.3. Tuples

A tuple consists of a list of values enclosed within parentheses("()"). Similar to a list, a tuple can have values of different types. Unlike lists, neither can we add/remove values from tuples, nor can we update the values. Consider the same values from the above example of a list and store them as a tuple:

PYTHON CODE:

```
# Defining a tuple containing multiple type objects

tupleseq=(24,3.5,'C','Myarray',23)

print('Complete tuple : ',tupleseq)

print('Element at index 3: ',tupleseq)

print('All elements from index 1 onwards: ',tupleseq[1:])

print('All elements before index 3(excluding index 3):
',tupleseq[:3])

print('Elements from index 2 to index 4: ',tupleseq[2:5])
```

```
[ ]
# Defining a tuple containing multiple type objects
tupleseq=(24,3.5,'C','Myarray',23)
print('Complete tuple : ',tupleseq)
print('Element at index 3: ',tupleseq)
print('All elements from index 1 onwards: ',tupleseq[1:])
print('All elements before index 3(excluding index 3): ',tupleseq[:3])
print('Elements from index 2 to index 4: ',tupleseq[2:5])

Complete tuple :  (24, 3.5, 'C', 'Myarray', 23)
Element at index 3:  (24, 3.5, 'C', 'Myarray', 23)
All elements from index 1 onwards:  (3.5, 'C', 'Myarray', 23)
All elements before index 3(excluding index 3):  (24, 3.5, 'C')
Elements from index 2 to index 4:  ('C', 'Myarray', 23)
```

§ Arithmetic Operators on Tuples

Following are the operations performed on tuples:

- + will combine two tuples
- * will repeat a tuple

PYTHON CODE:

```
# Defining first tuple
tuple1=('a','b')

# Defining second tuple
tuple2=('c','d')

# Combining two tuples
print('Combined Tuple: ',tuple1+tuple2)

# Repeating a tuple 3 times
print('3 Times repeated Tuple: ',tuple1*3)
```

```
[ ]
# Defining first tuple
tuple1=('a','b')
# Defining second tuple
tuple2=('c','d')
# Combining two tuples
print('Combined Tuple: ',tuple1+tuple2)
# Repeating a tuple 3 times
print('3 Times repeated Tuple: ',tuple1*3)

Combined Tuple:  ('a', 'b', 'c', 'd')
3 Times repeated Tuple:  ('a', 'b', 'a', 'b', 'a', 'b')
```

1.2.4. Dictionaries

Dictionaries are associative arrays or hash tables and consist of values in {key : value} form. In other words, a dictionary maps a single key to a single value. The key or value can be any data type. Dictionaries are declared using curly braces, and key-value pairs are defined inside. Dictionary container does not support arithmetic operations as lists and tuples do.

PYTHON CODE:

```
#the value before colon (:) is the key and value after
# colon(:) is the mapped value

mydict={1:'one',
      2:'two',
      3:'three',
      4:'four'}

# We access values from a dictionary using its specific key.
# For example,

print('Value against key 4:',mydict[4])
print('Value against key 2:',mydict[2])

#we can change the value of a key

mydict[2]='new value two'
print('New value against key 2:',mydict[2])

#we can add new key-value pair to the dictionary

mydict[7]='Seven'
print('Value of newly added key 7:',mydict[7])

#we can get all the keys and values of the dictionary

print('All keys of dictionary:',mydict.keys())
print('All values of dictionary:',mydict.values())
```

```
Value against key 4: four
Value against key 2: two
New value against key 2: new value two
Value of newly added key 7: Seven
All keys of dictionary: dict_keys([1, 2, 3, 4, 7])
All values of dictionary: dict_values(['one', 'new value two', 'three', 'four', 'Seven'])
```

2

Introduction to Numpy

Numpy is a package for scientific computing that is extremely efficient for mathematical computations, especially when we are dealing with matrices. That is why it is widely used across the globe for deep learning and machine learning data handling and manipulation. Numpy provides a wide range of functionalities. However, we will focus only on the basic functions.

2.1. Numpy Arrays

A numpy array is actually a grid of the same type of values. The array is indexed by a tuple of non-negative integers. The shape of an array is a tuple of integers giving the size of the array along each dimension. We can initialize a numpy array from a python list or a list of lists. Consider the following functions provided by Numpy to initialize Numpy arrays of different types:

- For a matrix filled with zeros: **zeros(shape)** where the shape is a tuple of integers
- For a matrix filled with ones: **ones(shape)** where the shape is a tuple of integers

- For a random matrix with values between 0 and 1: **random.rand()**
- For a random integers matrix: **random.randint (low,high,shape)**, where low represents the lowest value, high represents the highest possible value for the array and shape is a tuple of integers
- We can get the shape (dimensions) of a numpy array using "shape" property: **nparray.shape**. This function returns (height, width, channel) for a 3-dimensional array.

PYTHON CODE:

```
# Import numpy library
import numpy as np

# A numpy array of shape (5 x 5) and filled with all zeros

zeros=np.zeros(shape=(5,5))
print('zeros array:\n',zeros)

# A numpy array of shape (5 x 5) and filled with all ones

ones=np.ones(shape=(5,5))
print('\nones array:\n',ones)

# Numpy random.rand will have values between 0 and 1
# A random numpy array of shape (5 x 5) with values between 0
and 1

randarr=np.random.rand(5,5)
print('\nrandom array:\n',randarr)

# Numpy random int
# A random integer numpy array of shape (5 x 5) with values
between -3 and 3

randintarr=np.random.randint(-3,3,(5,5))
print('\nrandom int array: \n',randintarr)
```

```
zeros 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.]]

ones 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.]]

random array:
 [[0.70464511 0.98078331 0.77120195 0.94710599 0.68875285]
 [0.32763375 0.55604462 0.4949186  0.56866126 0.07202512]
 [0.41047261 0.14642314 0.33417508 0.47632654 0.73085607]
 [0.72340968 0.94077419 0.58865108 0.75026618 0.11950349]
 [0.47626131 0.99350592 0.06698833 0.60489666 0.93958879]]

random int array:
 [[-1  2 -1 -3  0]
 [-3  2 -3 -3  2]
 [-1 -3 -1 -3  2]
 [-3 -1 -1 -3  2]
 [ 0  2  0 -3  0]]
```

2.1.1. Array Slices

Arrays indices work just as lists in Python. We access the element of an array by providing its position in each dimension. We will look into array slices here. For example, consider the following array and its slices:

PYTHON CODE:

```
# randomint array
A= np.random.randint(-5,5,(5,5))
print('Random array: \n',A)

print('\nFirst row: ',A[0,:])
print('Third row: ',A[2,:])
print('Third row, third element onwards: ',A[2,2:])

print('\nSecond column: ',A[:,1])
print('Fourth column: ',A[:,3])
print('Fourth column, third element onwards: ',A[2:,3])
```

```
[ ]
1  # randomint array
2  A= np.random.randint(-5,5,(5,5))
3  print('Random array: \n',A)
4
5  print('\nFirst row: ',A[0,:])
6  print('Third row: ',A[2,:])
7  print('Third row, third element onwards: ',A[2,2:])
8
9  print('\nSecond column: ',A[:,1])
10 print('Fourth column: ',A[:,3])
11 print('Fourth column, third element onwards: ',A[2:,3])
```

```
Random array:
 [[-3 -3  1  0  1]
 [-4  4  0  4  2]
 [-2 -4 -4 -4 -3]
 [-3 -1  2  2  3]
 [ 4 -1  0  1 -3]]

First row:  [-3 -3  1  0  1]
Third row:  [-2 -4 -4 -4 -3]
Third row, third element onwards:  [-4 -4 -3]

Second column:  [-3  4 -4 -1 -1]
Fourth column:  [ 0  4 -4  2  1]
Fourth column, third element onwards:  [-4  2  1]
```

2.1.2. Array Manipulations

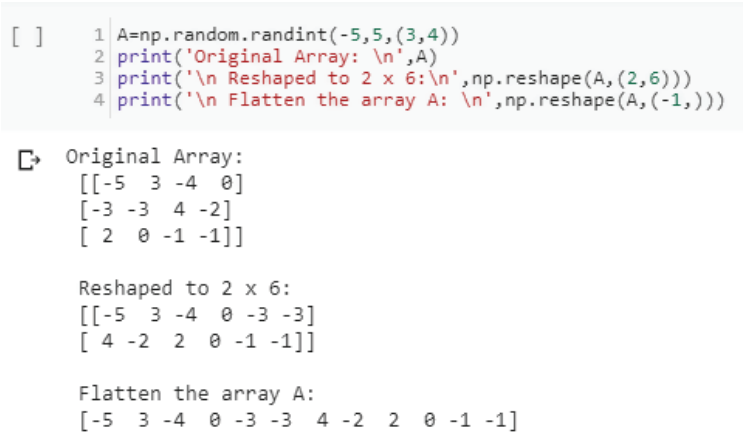
Numpy provides a function to transform a numpy array to a new shape:

`reshape(A,shape)`

- A: Numpy array to be reshaped
- shape: New shape to transform numpy array into We can reshape a numpy array to a particular shape. A value of −1 means to infer the value from remaining dimensions and original array shape

PYTHON CODE:

```
A=np.random.randint(-5,5,(3,4))
print('Original Array: \n',A)
print('\n Reshaped to 2 x 6:\n',np.reshape(A,(2,6)))
print('\n Flatten the array A: \n',np.reshape(A,(-1,)))
```



2.2. Arithmetic Operations

Arithmetic operations can be seamlessly applied to Numpy arrays as on numbers. We can use the same arithmetic notations for addition and subtraction. The functionality of the multiplication sign is a little different.

- For element-wise multiplication of two tensors, use **(*)** sign
- For multiplication of two tensors, use **np.dot** function

2.2.3. Binary Operations

We will first cover arithmetic operations performed on two numpy arrays:

PYTHON CODE:

```
# Addition
print('Ones + Randomarray: \n',ones+randarr)

#Subtraction
print('\nRandintarray - Randomarray: \n',
randintarr-randarr)

#Multiplication
print('\nRandintarray * Randomarray: \n',
np.dot(randintarr,randarr))

#Element wise Multiplication
print('\n Element wise multiply: randintarray and
randomarray:\n',randintarr*randarr)
```

```
Ones + Randomarray:
 [[1.70464511 1.98078331 1.77120195 1.94710599 1.68875285]
 [1.32763375 1.55604462 1.4949186  1.56866126 1.07202512]
 [1.41047261 1.14642314 1.33417508 1.47632654 1.73085607]
 [1.72340968 1.94077419 1.58865108 1.75026618 1.11950349]
 [1.47626131 1.99350592 1.06698833 1.60489666 1.93958879]]

Randintarray - Randomarray:
 [[-1.70464511  1.01921669 -1.77120195 -3.94710599 -0.68875285]
 [-3.32763375  1.44395538 -3.4949186  -3.56866126  1.92797488]
 [-1.41047261 -3.14642314 -1.33417508 -3.47632654  1.26914393]
 [-3.72340968 -1.94077419 -1.58865108 -3.75026618  1.88049651]
 [-0.47626131  1.00649408 -0.06698833 -3.60489666 -0.93958879]]

Randintarray * Randomarray:
 [[-2.63007926 -2.83743978 -1.88149306 -2.53690854 -1.63406917]
 [-3.90779208 -3.10484085 -3.95827044 -4.17398026 -2.59410944]
 [-3.3157254  -3.63065104 -4.2221094  -4.17042152 -0.1150172 ]
 [-4.06974812 -4.48012842 -4.77467609 -4.92731097 -1.34847266]
 [-1.51496154 -1.71023333 -0.77611604 -1.11347602 -0.21446024]]

 Element wise multiply: randintarray and randomarray:
 [[-0.70464511  1.96156662 -0.77120195 -2.84131796  0.        ]
 [-0.98290125  1.11208924 -1.4847558  -1.70598378  0.14405024]
 [-0.41047261 -0.43926943 -0.33417508 -1.42897961  1.46171215]
 [-2.17022904 -0.94077419 -0.58865108 -2.25079854  0.23900699]
 [ 0.          1.98701184  0.         -1.81468998  0.        ]]
```

PYTHON CODE:

```
# A random numpy array with size (3 x 3) and values between -5
and 5
randintarr=np.random.randint(-5,5,(3,3))

print('randint array:\n',randintarr)
print('\n 2 * randint: \n',2*randintarr)
print('\n (1/2) * randint: \n',randintarr/2)
```

```
randint array:
[[ 1  2  0]
 [ 3 -1  3]
 [-2  3  1]]

2 * randint:
[[ 2  4  0]
 [ 6 -2  6]
 [-4  6  2]]

(1/2) * randint:
[[ 0.5  1.   0. ]
 [ 1.5 -0.5  1.5]
 [-1.   1.5  0.5]]
```

2.2.4. Operations on a Single Array

Now, we will learn about functions performed on a single numpy array.

Sum Function

Numpy provides the sum function to calculate the sum of an array (all elements sum or axis-wise sum):

`numpy.sum(A,axis)`

- A: Numpy array
- Axis: Axis along which to calculate the sum. If None or not defined, it will calculate the sum of all the elements of an array. For example, if axis = 0, the sum will be

calculated for each column, and if axis = 1, the sum will be calculated for each row.

PYTHON CODE:

```
# Create a random numpy array of shape (3 x 4) with values
between -5 and 5
A=np.random.randint(-5,5,(3,4))
print('A:\n',A)

# Sum of all values of array A
print('\nSum of all elements: ',np.sum(A))

# Sum of all values of array A along the axis 0
print('Sum along axis 0: ',np.sum(A,axis=0))

# Sum of all values of array A along the axis 1
print('Sum along axis 1: ',np.sum(A,axis=1))
```

```
A:
 [[ 1 -1 -3  1]
 [-4 -5  4 -5]
 [-4 -5  2  0]]

Sum of all elements:  -19
Sum along axis 0:  [ -7 -11   3  -4]
Sum along axis 1:  [ -2 -10  -7]
```

Min Function

Numpy provides the amin function to calculate the minimum value in an array (for all elements or across an axis):

`numpy.amin(A,axis)`

- A: Numpy array
- Axis: Axis along which to find the minimum. If None or not defined, it will find a minimum of all elements in the array.

Similarly, other main functions in numpy are:

- **numpy.amax(A,axis)**: To find the maximum value in an array across an axis
- **numpy.mean(A,axis)**: To find mean value in an array across an axis
- **numpy.median(A,axis)**: To find median value in an array across an axis

In all of the above functions, if the axis is not provided, then the function is performed on the complete matrix (all elements).

PYTHON CODE:

```
A=np.random.randint(low=0,high=255,size=(5,5))
print('A:\n',A)
print('\nMinimum of array  :',np.amin(A))
print('Maximum of array  :',np.amax(A))
print('Mean of array     :',np.mean(A))
print('Median of array   :',np.median(A))

print('\nMinimum of array at axis=0  :',np.amin(A,axis=0))
print('Maximum of array at axis=0  :',np.amax(A,axis=0))
print('Mean of array at axis=0     :',np.mean(A,axis=0))
print('Median of array at axis=0   :',np.median(A,axis=0))

print('\nMinimum of array at axis=1  :',
np.amin(A,axis=1))
print('Maximum of array at axis=1  :',np.amax(A,axis=1))
print('Mean of array at axis=1     :',np.mean(A,axis=1))
print('Median of array at axis=1   :',np.median(A,axis=1))
```

```
A:
 [[142 123 165 243  36]
 [ 94 110 199 159 179]
 [ 63  19 215  46   5]
 [153  33 197 247 240]
 [247  84  47 168  14]]

Minimum of array  : 5
Maximum of array  : 247
Mean of array     : 129.12
Median of array   : 142.0

Minimum of array at axis=0  : [63 19 47 46  5]
Maximum of array at axis=0  : [247 123 215 247 240]
Mean of array at axis=0     : [139.8  73.8 164.6 172.6  94.8]
Median of array at axis=0   : [142.  84. 197. 168.  36.]

Minimum of array at axis=1  : [36 94  5 33 14]
Maximum of array at axis=1  : [243 199 215 247 247]
Mean of array at axis=1     : [141.8 148.2  69.6 174.  112. ]
Median of array at axis=1   : [142. 159.  46. 197.  84.]
```

2.3. Linear Algebraic Operations

We can perform operations like the inverse of a matrix, eigendecomposition, and transpose of a matrix in Python. For example, transpose of a matrix can be calculated as:

PYTHON CODE:

```
# A simple .T attribute of a numpy array returns its transpose

print('Transpose of A: \n',A.T)
```

```
Transpose of A:
 [[142  94  63 153 247]
 [123 110  19  33  84]
 [165 199 215 197  47]
 [243 159  46 247 168]
 [ 36 179   5 240  14]]
```

To calculate a pseudo-inverse of a matrix:

PYTHON CODE:

```
# Numpy's pinv function from linalg module, calculates pseudo
inverse.

print('Pseudo Inverse of A:\n',np.linalg.pinv(A))
```

```
Pseudo Inverse of A:
 [[-4.98607093e-03  4.75785527e-04  2.03728752e-03 -1.23403086e-05
   6.22201290e-03]
 [ 9.97999686e-04  9.91716571e-03 -3.28556840e-03 -7.54571462e-03
   1.16419276e-03]
 [-6.95311802e-05  1.81524255e-04  4.94213177e-03 -1.58412531e-04
  -1.19152646e-03]
 [ 7.21143908e-03 -6.07890818e-03 -2.54474134e-03  3.70817044e-03
  -3.48031722e-03]
 [-4.32330395e-03  4.44028495e-03 -2.28470868e-03  1.52577425e-03
   4.33261386e-04]]
```

2.4. Other Operations

A few more functions from Numpy that you need to understand are:

- tile
- concatenate
- dstack
- unique

2.4.1. Tile

Numpy's tile function replicates an array the number of times and dimensions given by reps:

`numpy.tile(A,reps)`

- A: Numpy array
- reps: Number of times A will be repeated along each dimension

Consider the following example where we repeat an array of 1 x 5 to make multiple rows of same values (tiling in dimension 0)

PYTHON CODE:

```
# A simple .T attribute of a numpy array returns its transpose
print('Transpose of A: \n',A.T)
# Generate a random numpy array of size (1 x 5) with values
between -5 and 5

A= np.random.randint(low=-5,high=5,size=(1,5))

print('A:\n',A)
print('\nA shape: ',A.shape)
tiled=np.tile(A,(5,1))

# Tile(replicate) A, 5 times in dimension 0 and 1 times in
dimension 1

print('\nTiled 5 times row wise(dimension 0) and 1 times
column wise(dimension 1):\n',tiled)

tiled=np.tile(A,(5,2))
print('\nTiled 5 times row wise(dimension 0) and 2 times
column wise(dimension 1):\n',tiled)
```

```
A:
 [[-3 -3  3  2 -4]]

A shape:  (1, 5)

Tiled 5 times row wise(dimension 0) and 1 times column wise(dimension 1):
 [[-3 -3  3  2 -4]
 [-3 -3  3  2 -4]
 [-3 -3  3  2 -4]
 [-3 -3  3  2 -4]
 [-3 -3  3  2 -4]]

Tiled 5 times row wise(dimension 0) and 2 times column wise(dimension 1):
 [[-3 -3  3  2 -4 -3 -3  3  2 -4]
 [-3 -3  3  2 -4 -3 -3  3  2 -4]
 [-3 -3  3  2 -4 -3 -3  3  2 -4]
 [-3 -3  3  2 -4 -3 -3  3  2 -4]
 [-3 -3  3  2 -4 -3 -3  3  2 -4]]
```

2.4.2. Concatenate

Numpy's concatenate function joins a sequence of arrays in a given dimension:

`numpy.concatenate(A,axis)`

- A: A tuple of numpy arrays to be concatenated (A1,A2,A3...)
- axis: The axis number to join the arrays in

Consider a simple example below, where three numpy arrays of the same dimensions are combined together along rows (axis = 0).

PYTHON CODE:

```
# A random array of 1 x 5 (row x column) with values between
-5 and 5

A= np.random.randint(low=-5,high=5,size=(1,5))

# A random array of 1 x 5 (row x column) with values between
-5 and 5

B= np.random.randint(low=-5,high=5,size=(1,5))

# A random array of 1 x 5 (row x column) with values between
-5 and 5

C= np.random.randint(low=-5,high=5,size=(1,5))

print('A:',A)
print('B:',B)
print('C:',C)

# Concatenate 3 arrays along the 0th axis

concatenated=np.concatenate((A,B,C),axis=0)
print('\nConcatenated:\n',concatenated)
```

```
A: [[ 0  4 -4 -2  4]]
B: [[ 4  1 -1  1  1]]
C: [[ 3 -4 -5  4 -1]]

Concatenated:
 [[ 0  4 -4 -2  4]
 [ 4  1 -1  1  1]
 [ 3 -4 -5  4 -1]]
```

2.4.3. Dstack

Numpy's dstack function stacks a sequence of arrays in depth dimension. We will be using this function to deal with 3-channel (RGB) images.

`numpy.dstack(A)`

- A: A tuple of numpy arrays to be stacked (A1,A2,A3...)

Consider the example below, where three numpy arrays of the same dimensions are stacked together:

PYTHON CODE:

```
R=np.random.randint(0,255,size=(24,24))
G=np.random.randint(0,255,size=(24,24))
B=np.random.randint(0,255,size=(24,24))
print('A shape:',R.shape)
print('B shape:',G.shape)
print('C shape:',B.shape)
stacked=np.dstack((R,G,B))
print('Stacked shape:',stacked.shape)
```

2.4.4. Unique

Numpy's unique function returns unique values in an array.

`numpy.unique(A,return_index)`

- A: A numpy array
- return_index: If true, returns the indices of unique elements' first occurrence in the original array

PYTHON CODE:

```
A = np.random.randint(low=-2,high=2,size=(15,))

print('A:\n',A)
print('\nUnique from array:',np.unique(A))

uniquevals,indices=np.unique(A,return_index=True)

print('\nUnique from array with return indices: ',uniquevals,'
- Indices in original array:',indices)
```

```
A:
 [ 1 -2 -2  0 -2  0  0  1 -1 -1  1  0  1 -2  1]

Unique from array: [-2 -1  0  1]

Unique from array with return indices:  [-2 -1  0  1]  - Indices in original array: [1 8 3 0]
```

3

Introduction to Pandas

Pandas is an open-source python project which provides highly efficient data manipulation and analysis tool by taking advantage of its powerful data structures. In this chapter, we will learn about the basics of pandas. This library provides the dataframe data structure for data structuring and organization. Dataframe is a 2D data structure that arranges data in rows and columns with the flexibility to allow different data types of columns.

3.1. Creating a DataFrame

A pandas dataframe can be created using the constructor:

```
pandas.DataFrame( data, index, columns, dtype)
```

- data: It can be a Python list, numpy array, dictionary, or any other dataframe.
- index: It is used for indexing rows for the resulting dataframe.
- columns: It is used for column labels.
- dtype: To define the data type of each column.

3.1.1. Creating Dataframe Using Python List:

Let's create some dataframes from different types of data. First, we will create a dataframe using a simple Python list:

PYTHON CODE:

```
# Import pandas library

import pandas as pd

# Defined a List of lists

_list = [[‘car’,’truck’],[‘20000’,’30000’]]

print (‘Python list\n’, _list)
print(‘\nDataframe:’)

# create a dataframe from the list of lists

df = pd.DataFrame(_list)
print (df)
```

```
Python list
 [['car', 'truck'], ['20000', '30000']]

Dataframe:
       0      1
0    car  truck
1  20000  30000
```

As you can see, the indexes and columns are numbered from 0 to 1. Since we did not pass any index and column labels, the dataframe is by default indexed and labeled using np.arrange(n).

3.1.2. Creating DataFrame from Python Dictionary

PYTHON CODE:

```
# A dictionary with keys defining column headings and arrays
define their corresponding columns

dictionary = {'Name':['James', 'Jonathan', 'Ahmed',
'Jacque'],'Age':[28,34,29,42]}
print ('Dictionary\n',dictionary)

# Convert dictionary to dataframe

df = pd.DataFrame(dictionary)
print (df)
```

```
Dictionary
 {'Name': ['Hassan', 'Muhammad', 'Ahmed', 'Ali'], 'Age': [28, 34, 29, 42]}
      Name  Age
0    Hassan   28
1  Muhammad   34
2     Ahmed   29
3       Ali   42
```

3.2. Accessing DataFrame Elements

Now, we are going to see how we can extract rows, columns, or a combination of both from dataframes. The general rule for extracting a subset of a DataFrame is given below.

```
DataFrame.loc[startrow:endrow, startcolumn:endcolumn]
```

Let's see a few examples of how we can put this syntax to use.

PYTHON CODE:

```
# Create a dataframe

_list = [['Car',10000,100],['Truck',12000,200],['SUV',13000,300]]

# Assign Column names to dataframe

df = pd.DataFrame(_list,columns=['Vehicle','Price','Engine'])

print (df)
```

	Vehicle	Price	Engine
0	Car	10000	100
1	Truck	12000	200
2	SUV	13000	300

We will use the general rule stated above to extract a subset of rows and columns from this dataset.

PYTHON CODE:

```
# Extract 0 - 1 rows and Vehicle : Price columns

df.loc[0:1,'Vehicle':'Price']
```

	Vehicle	Price
0	Car	10000
1	Truck	12000

3.2.1. Accessing All Values in a Column

We can also access all the rows for any column, as demonstrated in the following cell. We use (:) to specify all values in that dimension (in our case, dimension 0) and the column name to specify the column to extract values from.

PYTHON CODE:

```
# Extract all values of column 'Vehicle'

  df.loc[:,'Vehicle']
```

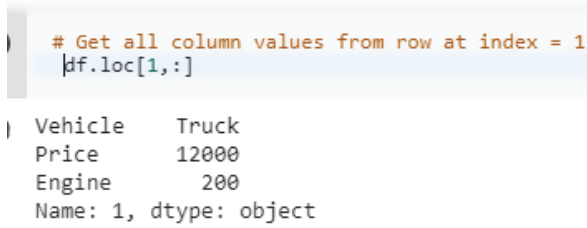
```
0      Car
1    Truck
2      SUV
Name: Vehicle, dtype: object
```

3.2.2. Accessing All Values in a Row

Similarly, we can access all values in a column for a specific row.

PYTHON CODE:

```
# Get all column values from row at index = 1

  df.loc[1,:]
```

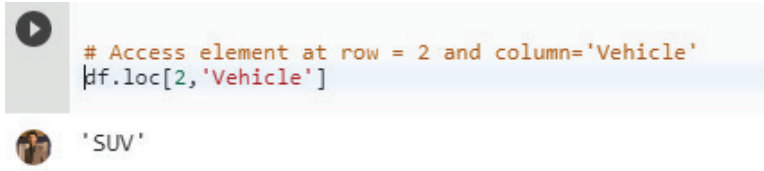


3.2.3. Accessing a Single Element from Array

We can access a particular entry in the array by specifying the row and column value of the element position.

PYTHON CODE:

```
# Access element at row = 2 and column='Vehicle'

 df.loc[2,'Vehicle']
```



3.3. Updating Elements

We can update an element in a dataframe just like we update elements in a Python list. To update an element, we need to

locate it by specifying its position, the row-column pair, and then assigning the value for that position.

PYTHON CODE:

```
print('Before Updating element: ')
print(df)

# update second row of column 'Vehicle' to 'sedan'

df.loc[2,'Vehicle'] = 'sedan'
print ('\nAfter updateing the element: \n',df)
```

```
print('Before Updating element: ')
print(df)

# update second row of column 'Vehicle' to 'sedan
df.loc[2,'Vehicle'] = 'sedan'
print ('\nAfter updateing the element: \n',df)
```

```
Before Updating element:
  Vehicle  Price  Engine
0     Car  10000     100
1   Truck  12000     200
2     SUV  13000     300

After updateing the element:
   Vehicle  Price  Engine
0     Car  10000     100
1   Truck  12000     200
2   sedan  13000     300
```

4

Introduction to Matplotlib

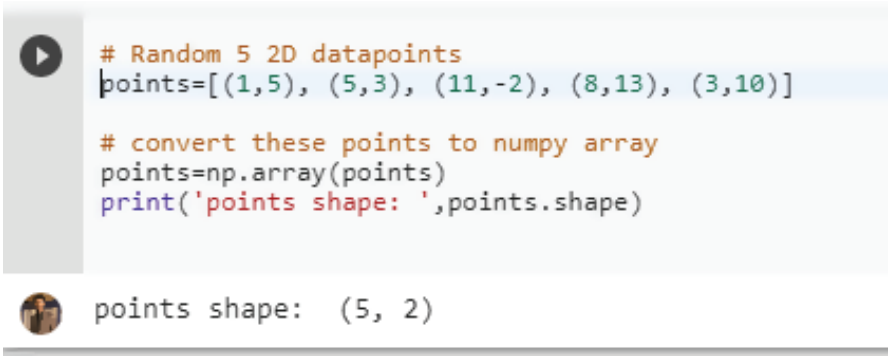
To visualize data distributions, losses, and accuracies or even to display images, we use matplotlib. It is very intuitive and provides an easy way to draw plots, graphs, histograms, bar charts, or scatter graphs.

4.1. Drawing 2D Points

We will start with the fundamental part: drawing points on 2D cartesian coordinates. Consider these five points and let's draw them: (1,5), (5,3), (11,-2), (8,13), (3,10)

PYTHON CODE:

```
# Random 5 2D datapoints

points=[(1,5), (5,3), (11,-2), (8,13), (3,10)]

# convert these points to numpy array

points=np.array(points)
print('points shape: ',points.shape)
```



The shape of the points tells us that we have five points, and each point has two coordinates. We know that the first coordinates are x, and the second coordinates are y, so we will separate them.

PYTHON CODE:

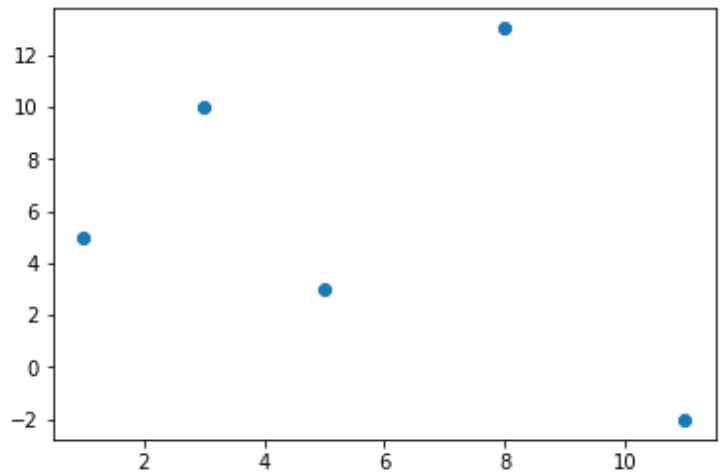
```
# All values in first column of points array

 x=points[:,0]

# All values in second column of points array

 y=points[:,1]
```

Following are the steps to visualize our points:

1. Import the pyplot module from the Matplotlib library
2. Initialize a figure using figure() function
3. Plot these points using scatter function
4. Call the show() function to display the resulting plot

PYTHON CODE:

```
# Import pyplot module of matplotlib library

 import matplotlib.pyplot as plt

# Initialize a figure
 plt.figure()

# scatter function will take two inputs: x values and their
corresponding y values

 plt.scatter(x,y)
 plt.show()
```




```
scatter(x, y, s=None, c=None, marker=None, cmap=None, norm=None,
vmin=None, vmax=None, alpha=None, linewidths=None, verts=None,
edgecolors=None, *, plotnonfinite=False, data=None, **kwargs)
```

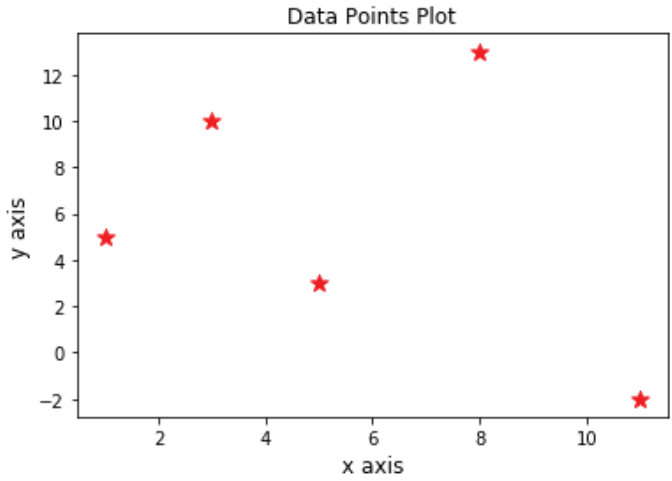
- x,y: these are the x and y coordinates of data points
- s: the marker size or size of the points
- c: a color or a sequence of colors for data points
- marker: marker style which is used to represent the points

- cmap: to use some of the available color maps for data points
- alpha: a value between 0 and 1. 0 means fully transparent, and 1 means opaque.
- edgecolors: a color or a sequence of colors for the marker's edge
- **kwargs: To provide more arguments as a dictionary if required.

Now that we have our first plot, we will add further details and attributes to it in the cell below.

PYTHON CODE:

```
import matplotlib.pyplot as plt

# Initialize a figure
plt.figure()

# Set title of figure
plt.title('Data Points Plot')

# plot data points
plt.scatter(x,y,s=100,marker='*',c='r')

# This will label X - axis of the plot
plt.xlabel('x axis',fontsize=12)

# This will label Y - axis of the plot
plt.ylabel('y axis',fontsize=12)

# To show the plot
plt.show()
```



4.2. Line Plotting

In Matplotlib, a line is a sequence of coordinates joined together. Consider an equation of the line: y = -13x + 7. Now, for a sequence of x values, we will have their respective y values for this line. To get a sequence of equidistant x values, Numpy provides the linspace function:

```
linspace(start,stop,num,endpoint=True)
```

- start: starting value of the range
- end: end value of the range
- num: number of equally distant samples in range (start,stop)
- endpoint: if True, the stop value is included in the range, and if False, stop value is excluded.

PYTHON CODE:

```
import numpy as np

# First, we generate a sequence of x values. 50 equally
distant values between -10 and 10

x = np.linspace(start=-10,stop=10,num=50)

# Get corresponding y values
y = -13 * x + 7
```

Now we have our x values and their corresponding y values. To draw a line, we will use the plot function from pyplot:

```
plot(x,y,fmt,label,linewidth)
```

- x: x values
- y: corresponding y values
- fmt: a format string. Here we define the color, marker, and border type of the line.
- label: a string value used as a label/name of the line. We can use this label to represent a line in plot legends.
- linewidth: a float value for the width of the line.

Pyplot provides two more functions specifically to draw horizontal and vertical lines.

For the horizontal line:

```
axhline(y,**kwargs)
```

- y: a float value on the y-axis to draw a horizontal line
- kwargs: to set the color, line style, line width, and other properties

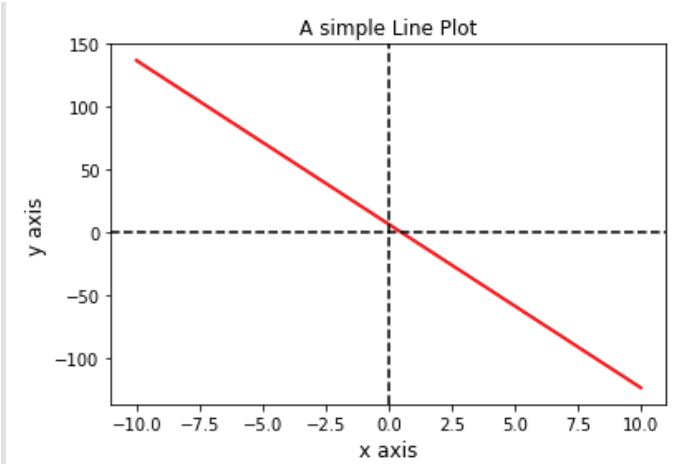
For the vertical line:

`axvline(x,**kwargs)`

- x: a float value on the x-axis to draw a vertical line
- kwargs: to set the color, line style, line width, and other properties

PYTHON CODE:

```
import matplotlib.pyplot as plt

# Initialize a figure
plt.figure()

# Set title of figure
plt.title('A simple Line Plot')

# Plotting the line
plt.plot(x,y,'r-',label='Line1',linewidth=2)

# This will label X - axis of the plot
plt.xlabel('x axis',fontsize=12)

# This will label Y - axis of the plot
plt.ylabel('y axis',fontsize=12)

# This will draw a black horizontal line at y = 0

plt.axhline(y=0,c='black',linestyle='--')

# This will draw a black vertical line at x = 0

plt.axvline(x=0,c='black',linestyle='--')

# To show the plot
plt.show()
```



We can draw multiple lines and points on a single plot. Consider the following example, where two lines and two sets of 2D points are drawn. To differentiate between lines and set of points, we can assign different colors to each of them, or we can use labels to caption in legends.

Legends:

To add a legend to our plot, we can call legend function from pyplot.

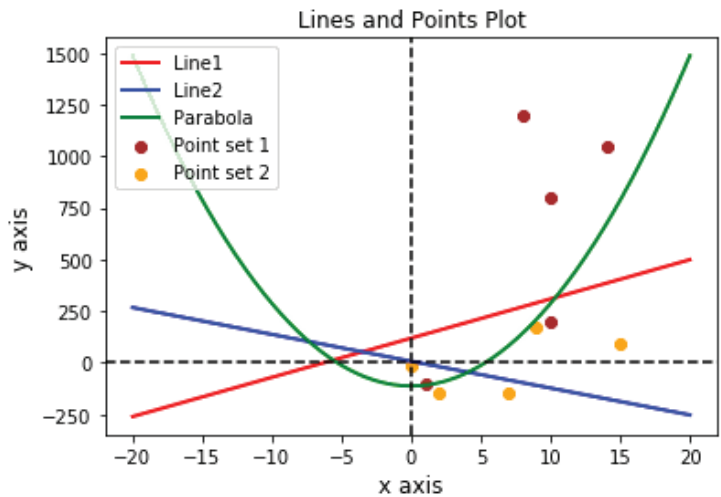
```
legend(loc, **kwargs)
```

- loc: This argument passed to legend function tells where to place the legend. There are 11 different positions available by default. We can pass the location using a number or a string, e.g., 'upper left' means to place the legend at the top left of the plot.

PYTHON CODE:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(start=-20,stop=20,num=1000)

# Defining lines and parabolas using equations
y1 = 19*x + 119
y2 = -13*x + 7
y3 = 4*x**2 - 113

# We will define two set of points

rand_points_set1=np.
array([[10,800],[8,1200],[14,1050],[1,-100],[10,200]])
rand_points_set2=np.
array([[9,173],[15,90],[2,-150],[0,-11],[7,-143]])

# Initialize a figure
plt.figure()

# Set title of figure
plt.title('Lines and Points Plot')

# Plotting first line
plt.plot(x,y1,'r-',label='Line1',linewidth=2)

# Plotting second line
plt.plot(x,y2,'b-',label='Line2',linewidth=2)

# Plotting a quadratic function

plt.plot(x,y3,'g-',label='Parabola',linewidth=2)
plt.scatter(rand_points_set1[:,0],rand_points_
set1[:,1],c='brown',label='Point set 1')
plt.scatter(rand_points_set2[:,0],rand_points_
set2[:,1],c='orange',label='Point set 2')
```

```
# This will label X - axis of the plot
plt.xlabel('x axis',fontsize=12)

# This will label Y - axis of the plot
plt.ylabel('y axis',fontsize=12)

# This will draw a black horizontal line at y = 0

plt.axhline(y=0,c='black',linestyle='--')

# This will draw a black vertical line at x = 0

plt.axvline(x=0,c='black',linestyle='--')

# Place legend at top left corner of the plot

plt.legend(loc='upper left')

# To show the plot
plt.show()
```



We can set a custom figure size (height and width). We can also set a limit on the x-axis and y-axis areas shown in the image.

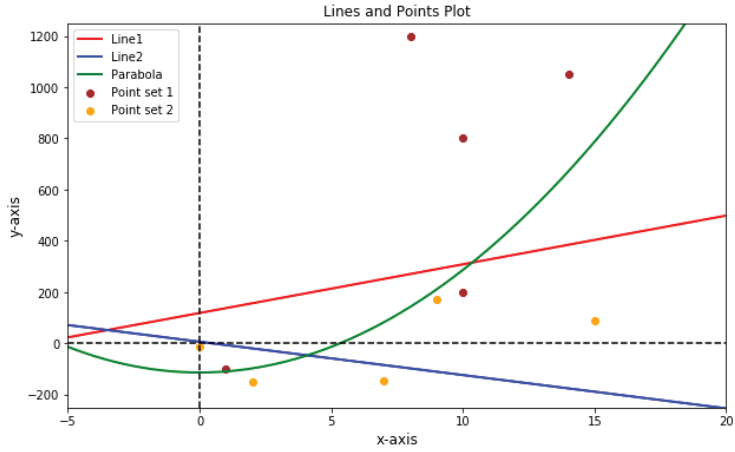
- To set the height and width of the figure, we can pass figsize tuple (height, width) in inches during figure initialization.
- To set the x and y-axis limit, we can call function xlim(minval,maxval) and ylim(minval,maxval) to define the range of each axis visible.

Consider the same example as above with increased height, width, and decreased x and y-axis visible area.

PYTHON CODE:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(start=-20,stop=20,num=1000)

y1 = 19*x + 119
y2 = -13*x + 7
y3 = 4*x**2 - 113

rand_points_set1=np.
array([[10,800],[8,1200],[14,1050],[1,-100],[10,200]])
rand_points_set2=np.
array([[9,173],[15,90],[2,-150],[0,-11],[7,-143]])

# Initialize a figure and set the figure size to 10 inches width
and 6 inches height

plt.figure(figsize=(10,6))

# Set title of figure
plt.title('Lines and Points Plot')

# Plotting first line
plt.plot(x,y1,'r-',label='Line1',linewidth=2)

# Plotting second line
plt.plot(x,y2,'b-',label='Line2',linewidth=2)
```

```
# Plotting a quadratic function
plt.plot(x,y3,'g-',label='Parabola',linewidth=2)

# Plottting points

plt.scatter(rand_points_set1[:,0],rand_points_
set1[:,1],c='brown',label='Point set 1')

plt.scatter(rand_points_set2[:,0],rand_points_
set2[:,1],c='orange',label='Point set 2')

# This will label X - axis of the plot
plt.xlabel('x-axis',fontsize=12)

# This will label Y - axis of the plot
plt.ylabel('y-axis',fontsize=12)

# This will draw a black horizontal line at y = 0
plt.axhline(y=0,c='black',linestyle='--')

# This will draw a black vertical line at x = 0
plt.axvline(x=0,c='black',linestyle='--')

# Only show x-axis from -5 to 20
plt.xlim(-5,20)

# Only show y axis from -250 to 1250
plt.ylim(-250,1250)

# Place legend at top left corner of the plot
plt.legend(loc='upper left')

# To show the plot
plt.show()
```



As you can see, our view area of the x-axis and y-axis has decreased (to a specific portion), and figure size has also increased.

4.3. Subplots

Sometimes, we want figures to be drawn side by side, for comparison, or for being relevant. Matplotlib provides this functionality as subplots. We can arrange images in a grid or in linear form, and each image is considered a subplot. Following is a simple example with two plots drawn side by side. We initialize subplots by calling "subplots" function and passing the number of rows and columns of images we want. The initialization returns two handles: one for complete figure and one for each axis. For example, as we create a figure with two subplots, it will return one handle for the whole figure, and two axes handle through which we can access each subplot.

Some functions which we discussed above, change as:

- To set the title of a subplot, we call "set_title()" on axes. To set the title of a complete figure, we call "subtitle" on

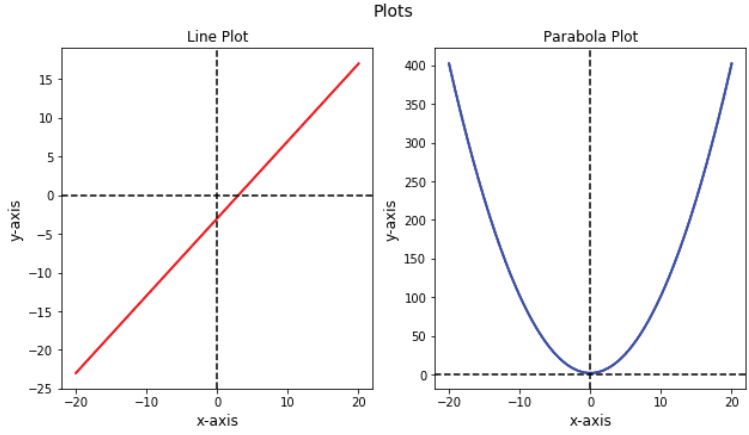
the figure handle returned by subplots function.

- We can access a subplot using its axes handle.
- Rather than simple xlabel() and ylabel() functions, we have set_xlabel() and set_ylabel() functions for subplots with same parameters.
- To set figure size, we use figure handle and call function set_size_inches and pass width and height values.

PYTHON CODE:

```
import matplotlib.pyplot as plt
import numpy as np

# Divide numbers between -20 and 20 in 1000 equally distant
values

x = np.linspace(start=-20,stop=20,num=1000)

# Define two plots (y1 and y2)
y1 = x - 3
y2 = x**2 + 2

# Define subplots
# subplots: 1 row and 2 column (2 images side by side)

fig,ax = plt.subplots(1,2)
fig.set_size_inches(10.25, 5.25)

# Set title of complete figure
fig.suptitle('Plots',fontsize=14)

# Set title of axes 0 or subplot 1
ax[0].set_title('Line Plot')

# Plotting first line
ax[0].plot(x,y1,'r-',linewidth=2)
```

```
for i in range(2):
    # This will label X - axis of both subplots
    ax[i].set_xlabel('x-axis',fontsize=12)

    # This will label Y - axis of both subplots
    ax[i].set_ylabel('y-axis',fontsize=12)

    # This will draw a black horizontal line at y = 0
    ax[i].axhline(y=0,c='black',linestyle='--')

    # This will draw a black vertical line at x = 0
    ax[i].axvline(x=0,c='black',linestyle='--')

# To show the figure
plt.show()
```

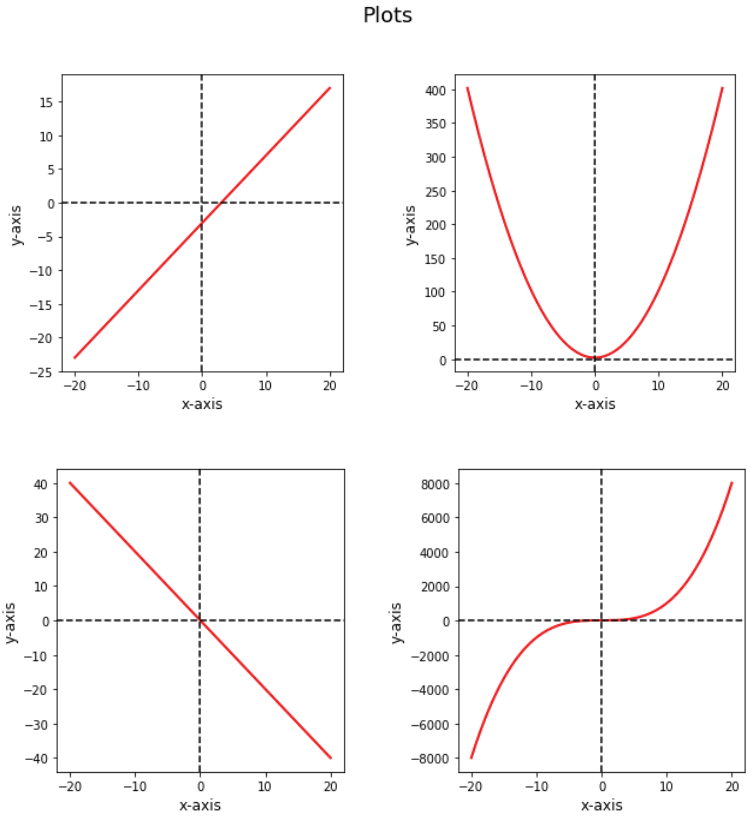


Look at another example with four subplots: As we now have four subplots shaped as a grid of 2 x 2, we need to access them using 2D indices.

PYTHON CODE:

```
import matplotlib.pyplot as plt
import numpy as np

# Divide numbers between -20 and 20 in 1000 equally distant
values

x = np.linspace(start=-20,stop=20,num=1000)

# Define 4 plots (y1,y2,y3,y4)
y1 = x - 3
y2 = x**2 + 2
y3 = -2*x
y4 = x**3 + 9

y=[y1,y2,y3,y4]

# subplots: 2 rows and 2 column (grid)
fig,ax = plt.subplots(2,2)

# Set height and width of figure in inches
fig.set_size_inches(10.25, 10.25)

# To adjust spacing and gap between subplots

plt.subplots_adjust(top=0.9,wspace=0.4,bottom=0.15)

# Set title of complete figure
fig.suptitle('Plots',fontsize=18)

# iterator to get y values

handle_idx=0
for i in range(2):
    for j in range(2):

        # Plot on axes
        ax[i,j].plot(x,y[handle_idx],'r-',linewidth=2)
```

```
        # This will label X - axis of subplot
        ax[i,j].set_xlabel('x-axis',fontsize=12)

        # This will label Y - axis of subplot
        ax[i,j].set_ylabel('y-axis',fontsize=12)

        # This will draw a black horizontal line at y = 0
        ax[i,j].axhline(y=0,c='black',linestyle='--')

        # This will draw a black vertical line at x = 0
        ax[i,j].axvline(x=0,c='black',linestyle='--')
        handle_idx+=1

# To show the figure
plt.show()
```



5

Introduction to OpenCV

OpenCV is an image processing library for Python. We will be using it to load images, process images, or manipulate images for preprocessing. In this chapter, we will learn about the basic functions from OpenCV, such as image loading, storing, and image manipulation.

5.1. Image Loading:

OpenCV's imread function reads an image file and returns a numpy array containing the pixel values. OpenCV reads the image in BGR format, which means the order of the color channels is (Blue, Green, Red) as compared to alternative format RGB (Red, Green, Blue). In other words, we find the Blue channel, Green channel, and Red channel at indices 0, 1, and 2, respectively.

```
imread(imagepath,loadflag)
```

- imagepath: Complete path of the image to load
- loadflag: To specify the form to load the image in. The three options are:
    - 1: To load a color image. Any transparency of the image will be neglected. This is the default option.

- 0: To load the image as a grayscale (one channel).
- -1: To load the image with transparency (alpha channel).

The imread function loads the image as a numpy array. Consider this example of reading an image:

PYTHON CODE:

```
# First, import opencv library (named cv2)
# importing opencv

import cv2

# Reading image

im=cv2.imread('images/catimage.jpeg',1)
print('Image shape:',im.shape)
```

The shape of the image is (667,1000,3): height = 667, width = 1000, channels = 3.

5.2. Image Display

To display the loaded image, we will use a function provided by pyplot from matplotlib.

`imshow(image,cmap)`

- image: Numpy array or image loaded using OpenCV
- cmap: This parameter is used when the passed image is a grayscale (1 channel); otherwise, it is ignored. It maps single scalar values in a grayscale image to a color range.

PYTHON CODE:

```
# Import pyplot module from matplotlib library

import matplotlib.pyplot as plt

# Call imshow function of pyplot module of library to display
the image

plt.imshow(im)

# Call .show() function to actually display the image as a
figure

plt.show()
```

As we can see, the image colors are not right. They are kind of inverted. The reason is that our loaded image is in BGR format, i.e., our channels are sequences as (Blue, Green, Red) while matplotlib requires images in RGB format. There are two ways to transform our BGR image to RGB format:

1. Reorder the stack of channels in our image using numpy dstack.
2. Use another function provided by OpenCV: cvtColor

We will convert our image from BGR to RGB using both ways and will see the results.

For the first point, we know that our image has dimensions (height, width, channel), and since it is a colored image, its channels are equal to 3. These channels are in BGR format. So, we will access each channel from the image and will restack them to get an image in RGB format. Then, we will use matplotlib to display our new image.

5.3. Conversion from BGR to RGB Using dstack

Since the Red channel will be at the last index of our BGR image, so to access it, we have to specify the last index: im[:,:,2] will select the red channel (in our case, 3rd or last channel) of BGR image. Similarly, im[:,:,1] will select the green channel (or 2nd channel) from the BGR image, and finally im[:,:,0] will select the blue channel (or 1st channel) from the BGR image. We will then stack them together in the RGB format.

PYTHON CODE:

```
import numpy as np

# Stacking channels to make RGB image

RGBformat=np.dstack((im[:,:,2],im[:,:,1],im[:,:,0]))

# Show image
plt.imshow(RGBformat)
plt.show()
```

5.4. Conversion from BGR to RGB Using cvtColor

Numpy's cvtColor function is used to change the colorspace of an image. We can use it to convert our color format.

`cvtColor(image,flag)`

- image: Numpy array image
- flag: specifying the type of conversion

There are several flags available for color space conversion, but we are only looking for one that converts BGR to RGB: cv2.COLOR_BGR2RGB

PYTHON CODE:

```
# OpenCV's cvtColor function converts image from BGR format to
RGB format

RGBimg=cv2.cvtColor(im,cv2.COLOR_BGR2RGB)

# Show RGB Image
plt.imshow(RGBimg)
plt.show()
```

5.4.1. Conversion from BGR to Grayscale Using cvtColor

In a similar way, we can convert our RGB image to HSV color space or grayscale. For conversion to grayscale:

PYTHON CODE:

```
# Using OpenCV's cvtColor function, we convert BGR formatted
image to Grayscale image.

grayscaleimg=cv2.cvtColor(im,cv2.COLOR_BGR2GRAY)

# we assign gray cmap because by default, cmap has value =
rc[image.cmap]

plt.imshow(grayscaleimg,cmap='gray')
plt.show()
```

5.5. Image Manipulation

Now, we will perform some operations on an image to understand basic image processing, which is usually done in data augmentation. For example, we will generate versions of an image with increased and decreased brightness as it can help the model learn features in low light or in bright conditions.

- To decrease brightness, we will divide the original pixel values by a number. This means all the pixel values will decrease by the same proportion and hence, we will get a darker image. Now consider a case when a pixel value is 230. Dividing it by three will give us a floating value. Pyplot's imshow function can take either (0 – 1) float values or (0 – 255) integer values. Thus, it will not be a valid image for matplotlib. To correct this, we first need to normalize the image. Which means, we will first bring all the pixel values in (0 – 1) range and will then perform division. This way, we will remain in (0 – 1) range.
- To increase brightness, we will multiply the pixel values by some real number. It is possible that some values will increase beyond the maximum value for a pixel, e.g., > 1.0 in our case. In this case, the imshow function automatically clips those pixel values to the highest possible (1.0).

PYTHON CODE:

```
# Read image (images/lenna.png) using imread function of
openCV

im=cv2.imread('images/lenna.png')

# Convert Image from BGR to RGB format
im=cv2.cvtColor(im,cv2.COLOR_BGR2RGB)

# Using subplots for image comparison
fig,ax=plt.subplots(1,3)
fig.set_size_inches(14.25, 8.25)

# Normalize Image:
im=im/255.0

# show image as a figure
ax[0].imshow(im)
ax[0].set_title('Original Image')

# This will divide all the pixel values by 3
# (so the brightness will be reduced by a factor of 3)

low_brightness=im/3
ax[1].imshow(low_brightness)
ax[1].set_title('Lowered Brightness')

# The pixel values increase by a factor of 1.5 and hence,
# the brightness of image increases by same factor

ax[2].imshow(high_brightness)
high_brightness = im*1.5
ax[2].set_title('Increased Brightness')

plt.show()
```

6

Introduction to Keras

This chapter is an introduction to Keras with an explanation of basic model building. We focus on individual functions to build each layer and their parameters, along with examples to see their results practically. More advanced concepts, e.g., fine-tuning and data augmentation, are explained in the future chapters.

6.1. Keras Library

Keras is a deep learning library. Its modular approach and simple design allow us to learn and implement deep learning models easily and quickly. Its design allows us to quickly prototype our models without worrying too much about intricate/complex inner details. It's a high-level library, so its syntax is easy to understand.

6.2. Keras Backend

As Keras is a high-level library, it does not perform low-level operations of tensor manipulation, e.g., tensor multiplication, addition, etc. Rather, it uses specialized third-party libraries to do that. These special libraries are said to be the backend engine of Keras. From various possible options of Keras

backend engines such as Theano, TensorFlow, and CNTK, we can select any one and use it with Keras.

First, we will use Keras backend with TensorFlow and understand tensor manipulations in it.

6.3. Tensors Manipulation

PYTHON CODE:

```
# Import keras backend to use

from keras import backend as K
```

We will define three tensors of 5 x 5 shape:

1. Filled with 0s
2. Filled with 1s
3. Filled with random values

A simple print statement is not used to show the values of the tensor. Keras backend provides get_value function to do that.

PYTHON CODE:

```
# Initialize a tensor of shape 5 x 5 with all zeros filled
zeros_tensor=K.zeros(shape=(5,5))

# Initialize a tensor of shape 5 x 5 with all ones filled
ones_tensor= K.ones(shape=(5,5))

# Initialize a random tensor of shape 5 x 5 with mean value =
0 and standard deviation = 3

rand_tensor=K.random_normal_
variable(shape=(5,5),mean=0,scale=3)

print('zeros tensor: ')
# Print zeros tensor
print(K.get_value(zeros_tensor))
```

```
print('\nones tensor: ')
# Print ones tensor
print(K.get_value(ones_tensor))

print('\nrandom tensor: ')
# Print random tensor
print(K.get_value(rand_tensor))

# Shape of the random tensor
print('\nShape of random tensor: ',rand_tensor.shape)
```

```
[[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.]]

ones tensor:
[[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.]]

random tensor:
[[-1.9247348   1.7806361   1.2424654   4.1707      0.03489759]
 [-1.0863552  -1.686175   -3.4082859  -1.1001399   4.92533   ]
 [-0.44365144 -2.1258113   2.6624944  -4.5818515   4.4941826 ]
 [-1.6052155  -1.7099472   2.3560743  -1.282958   -1.2331492 ]
 [ 0.13307902  1.4599998   3.871846    1.0848838  -0.18272676]]

Shape of random tensor:  (5, 5)
```

Now, we will add two tensors together, subtract two tensors, and multiply the values of two tensors to understand tensor operations. We can use the same arithmetic notations for addition and subtraction. The multiplication sign is a little confusing.

- For element-wise multiplication of two tensors, use '*' sign
- For multiplication of two tensors, use **K.dot** function

PYTHON CODE:

```
# Add random tensor with ones tensor

add_ones_rand = ones_tensor + rand_tensor

# Subtract ones tensor from random tensor

sub_rand_ones = rand_tensor - ones_tensor

# Element wise multiply ones and random tensor

mul_ones_rand = ones_tensor * rand_tensor

print('\nAddition : \n',K.get_value(add_ones_rand))
print('\nSubtraction : \n',K.get_value(sub_rand_ones))
print('\nMultiplication: \n',K.get_value(K.dot(ones_
tensor,rand_tensor)))
print('\nElement wise Multiply : \n',K.get_value(mul_ones_
rand))
```

```
[ ] Addition :
     [[-0.92473483  2.780636    2.2424655   5.1707      1.0348976 ]
     [-0.08635521 -0.686175   -2.4082859  -0.10013986  5.92533   ]
     [ 0.55634856 -1.1258113   3.6624944  -3.5818515   5.4941826 ]
     [-0.60521555 -0.7099472   3.3560743  -0.28295803 -0.23314917]
     [ 1.133079    2.4599998   4.871846    2.0848837   0.81727326]]

    Subtraction :
     [[-2.9247348   0.7806361   0.24246538  3.1707     -0.96510243]
     [-2.0863552  -2.6861749  -4.408286   -2.1001399   3.9253302 ]
     [-1.4436514  -3.1258113   1.6624944  -5.5818515   3.4941826 ]
     [-2.6052155  -2.709947    1.3560743  -2.282958   -2.233149  ]
     [-0.86692095  0.4599998   2.871846    0.08488381 -1.1827267 ]]

    Multiplication:
     [[-4.926878  -2.2812977  6.724594  -1.7093655  8.038535 ]
     [-4.926878  -2.2812977  6.724594  -1.7093655  8.038535 ]
     [-4.926878  -2.2812977  6.724594  -1.7093655  8.038535 ]
     [-4.926878  -2.2812977  6.724594  -1.7093655  8.038535 ]
     [-4.926878  -2.2812974  6.724594  -1.7093655  8.038534 ]]

    Element wise Multiply :
     [[-1.9247348   1.7806361   1.2424654   4.1707      0.03489759]
     [-1.0863552  -1.686175   -3.4082859  -1.1001399   4.92533   ]
     [-0.44365144 -2.1258113   2.6624944  -4.5818515   4.4941826 ]
     [-1.6052155  -1.7099472   2.3560743  -1.282958   -1.2331492 ]
     [ 0.13307902  1.4599998   3.871846    1.0848838  -0.18272676]]
```

6.4. Model Building

A deep learning model is a combination of different types of layers (e.g., convolution, pool, fully connected, dropout, activation). In Keras, these layers are used as modules and are put together using the "model" data structure.

In Keras, we can build our models in two ways:

- Using Sequential API
- Using Functional API

We use sequential API for developing simple models, e.g., with no skip connections, no acyclic graphs, or multiple outputs. Keras provides the 'Sequential' model to put together different modules as a linear stack. For complex architectures (e.g., containing skip connections, cyclic graphs), we use the functional API of Keras.

Following are the steps to build a model using Sequential API:

1. First, we define or initialize the model, which means declaring the model as an instance of Sequential Class. This allows us to stack different layers to be used as a model.
2. We add different layers to our model using ".add()" function.
3. Then, we compile our model, which means to configure the learning process and includes setting loss function and optimizer for the model or setting the learning rate for weight updates during backpropagation.
4. We train our model by calling the ".fit()" function on our model object. The parameters of fit function include

training data, validation data, number of epochs to train the model, and batch size.

5. Evaluating the model performance on the evaluation dataset. We can set a specific metric ('Accuracy', 'F1 score', 'Precision', etc.) to get results in required units.
6. Model Testing on our test set and visualizing the results.

6.5. Layers

Here, we will see how to define different modules (layers) in Keras, and later, we will learn to stack them using the Sequential API of Keras.

6.5.1. Convolutional Layer

We import CNN layers from Keras.layers.convolutional. We will consider the function for 2D convolution. It has several parameters, but we will only discuss the most important and relevant ones.

```
Conv2D(filters, kernel_size, strides=(1, 1), padding='valid',
activation=None, use_bias=True,data_format=None)
```

- filters: Filters actually tell the number of output channels because each filter is trained to look for a certain visual feature and outputs a single activation map for that feature. This means, if we have 32 filters, and apply Conv2D on an input tensor, we will get an output tensor with 32 channels.
- kernel_size: Kernel_size is an integer or a tuple that specifies the height and width of the convolution window. It is effectively our effective receptive field. A large kernel would mean that the kernel is looking at a

larger area of an image, while a small kernel size means that it focuses on a smaller area at one time.

- strides: Strides is an integer or a tuple that specifies the step for the convolution window along the height and width.
- padding: Possible padding options are 'valid' and 'same'. With the 'same' option, the output tensor has the same shape as the input tensor, as the image is padded before applying Conv2D.
- activation: The name of the activation function to be applied on the output of the layer. By default, no activation is applied.
- use_bias: True if the layer is to use bias vector and False if not.
- data_format: This is a string value that defines the ordering of input tensor dimensions. "channels_last" corresponds to inputs with shape (batch, height, width, channels) while "channels_first" corresponds to inputs with shape (batch, channels, height, width). By default, it will be the value set in ~/.keras/keras.json or "channels_last", if not set in the file.

This function takes a 4D tensor as input with shape (batch, channels, height, width).

See the example below of creating a Convolutional 2D layer:

PYTHON CODE:

```
# Import Conv2D submodule from keras' convolutional module

from keras.layers.convolutional import Conv2D

# Import keras as backend
from keras import backend as K

# Get a layer with padding = 'valid' and another layer with
padding = 'same'

convlayer_validpadding=Conv2D(1, (3,3), strides=(1, 1),
padding='valid', activation=None, use_bias=True,data_
format="channels_first")

convlayer_samepadding=Conv2D(1, (3,3), strides=(1, 1),
padding='same', activation=None, use_bias=True,data_
format="channels_first")
```

We will now initialize a random tensor of batch 1, channel 1, and height and width of 5. We will feed this tensor to our convolution layer and will look at the results. This will help us understand the convolution in Keras.

PYTHON CODE:

```
# A random tensor of uniform data distribution with shape =
(1 x 1 x 5 x 5) with lowest possible value = 0 and highest
possible = 255

randomtensor=K.random_uniform_
variable(shape=(1,1,5,5),low=0,high=255)
print('Input tensor: \n',K.get_value(randomtensor))

# Pass this random tensor with shape = (1 x 1 x 5 x 5)  to
convolutional layer
```

```
print('\nConvolution with valid padding: ')
outtensor=convlayer_validpadding(randomtensor)
print('Conv output: \n',K.get_value(outtensor))

# Pass this random tensor with shape = (1 x 1 x 5 x 5)  to
convolutional layer

print('\nConvolution with same padding: ')
outtensor=convlayer_samepadding(randomtensor)
print('Conv output: \n',K.get_value(outtensor))
```

```
Input tensor:
 [[[[ 65.76495   81.825356  15.02719   36.95554   32.877327]
   [209.33842   10.329335 159.12941  137.64525  116.365776]
   [240.34006  247.07956   15.511345  55.06701  144.0604  ]
   [204.52391  209.58885  156.07443  155.97627  201.62712 ]
   [ 42.551273  29.56971   41.977715  67.07628  118.80479 ]]]]

Convolution with valid padding:
Conv output:
 [[[[ -86.29592   -14.984547  -22.486248]
   [  62.06592   -38.12681  -119.31634 ]
   [ 161.76526   118.06208   -47.397507]]]]

Convolution with same padding:
Conv output:
 [[[[  53.43195    -27.072586   -27.5922      -4.6767797  -28.941502 ]
   [  97.45246   -147.2408      32.422558    -1.5409379  -69.10389  ]
   [ 136.74294   -132.81952   -147.16383    -45.018234  -103.6035   ]
   [ 169.78822   -172.38379   -129.70387     61.254646   -84.2743   ]
   [  39.110954  -116.932816   -85.53351    -20.845926  -126.87508  ]]]]
```

We can see from the output above that our output tensor has less width and height in valid padding while we have the same width and height in the case of the same padding.

6.5.2. Pooling Layer

We import pooling layers from Keras.layers. We will consider the function for MaxPooling2D as we are working with images, for now. The function parameters are the same for AveragePooling2D.

```
MaxPooling2D(pool_size=(2, 2), strides=None, padding='valid')
```

- pool_size: It is the window size that is actually an effective receptive field. It is an integer value or a tuple of two integers specifying the factors to downscale the input tensor in width and height. A pool_size of (2, 2) will halve the input in both spatial dimensions. For a single integer, the same window length will be used for both the dimensions.
- strides: It defines the steps in the width and height dimensions after each operation. It is an integer value or a tuple of two integers. If strides=None, the default value used will be pool_size.
- padding: It can either be "same" or "valid". If the height and width of input tensor are perfectly divisible by pool_size, we will have the same output for "same" and "valid" options. But the outputs will be different in other cases. We will mostly use the "valid" option in our deep learning models.
- data_format: Same as in the convolutional layer. It can be "channels_first", specifying that the input tensor will have channels as the first dimension or "channels_last", specifying that the input tensor will have channels as the last dimension.

PYTHON CODE:

```
# Import MaxPooling2D submodule from keras' layers module

from keras.layers import MaxPooling2D

# Import keras backend

from keras import backend as K

# Intialize two pooling layers: one with 'valid' padding and
the other with 'same' padding

poollayer_validpadding=MaxPooling2D(pool_size=(2, 2),
strides=None, padding='valid', data_format="channels_first")

poollayer_samepadding=MaxPooling2D(pool_size=(2, 2),
strides=None, padding='same',data_format="channels_first")

# Intiailize a random tensor with lowest value = 0 and highest
value = 255

randomtensor=K.random_uniform_
variable(shape=(1,1,5,5),low=0,high=255)
print('Input tensor: \n',K.get_value(randomtensor))

# Pass random tensor to pooling layer with valid padding

print('\nMaxpool with valid padding: ')
outtensor=poollayer_validpadding(randomtensor)
print('Conv output: \n',K.get_value(outtensor))

# Pass random tensor to pooling layer with same padding

print('\nMaxpool with same padding: ')
outtensor=poollayer_samepadding(randomtensor)
print('Maxpool output: \n',K.get_value(outtensor))
```

```
Input tensor:
 [[[[ 88.08201  254.96082  235.22987   32.803852 237.5966  ]
   [ 81.94276  105.93904   27.282171  45.248062 166.89757 ]
   [200.07713  245.2447   178.40118   28.157705  49.7963  ]
   [ 37.316795 183.24231   39.885426  98.707184  84.3887  ]
   [159.07239   17.028164 120.70368   52.929123  69.96987 ]]]]

Maxpool with valid padding:
Conv output:
 [[[[254.96082 235.22987]
   [245.2447  178.40118]]]]

Maxpool with same padding:
Maxpool output:
 [[[[254.96082 235.22987 237.5966 ]
   [245.2447  178.40118  84.3887 ]
   [159.07239 120.70368  69.96987]]]]
```

6.6. Activation

Activations are applied on the output of our model layers. Activations can be applied through the activation layer provided by Keras.layers.core submodule of Keras or activation parameter (or argument) supported by each layer. The activations are also provided through Keras.activations submodule.

We only specify the name of the activation function as the parameter:

```
Activation(activation)
```

- activation: We can choose from several available activation functions: ‘relu’, ‘softmax’, ‘tanh’, ‘sigmoid’, ‘exponential’, ‘linear’, and more.

The following are examples of two activation functions: ReLU and Softmax. For softmax, we need to define the axis along which to apply the activation.

Keras also provides some advanced activation functions which can learn parameters, but we will not study this advanced concept now.

PYTHON CODE:

```
# Import Activation submodule from keras layers.core module

from keras.layers.core import Activation

# Import keras backend
from keras import backend as K

# A random uniform distribution tensor with lowest possible
value = -2 and highest possible = 5

randomtensor=K.random_uniform_variable(shape=(1,1,5,5),low=-
2,high=5)

# Intialize relu activation

reluactivation=Activation('relu')
print('\n\t\t\tRELU ACTIVATION')
print('Input tensor: \n',K.get_value(randomtensor))

# Pass random tensor to relu activation to see its

effectsouttensor=reluactivation(randomtensor)
print('\nRelu Activation output: \n',K.get_value(outtensor))

print('-'*70)
print('\n\t\t\tSoftmax Activation')

# random uniform distribution tensor of shape (1 x 1 x 1 x 6),
lowest possible value = -2 and highest possible = 5

randomtensor=K.random_uniform_variable(shape=(1,1,1,6),low=-
2,high=5)
print('Input tensor: \n',K.get_value(randomtensor))
```

```
# Initializing softmax activation

softmaxactivation=Activation('softmax')

# Passing random tensor to softmax activation layer

outtensor=softmaxactivation(randomtensor)
print('\nSoftmax Activation output: \n',K.get_
value(outtensor))
```

```
                        RELU ACTIVATION
Input tensor:
 [[[[ 2.0705428   4.8187284   1.7726908   4.627082   -1.7312672 ]
   [ 4.862919    4.722542   -0.840021    4.540087    1.2682633 ]
   [-0.36543572  3.1376042   2.9521737   2.066959   -1.3211952 ]
   [ 0.13226175  1.7998643   2.1514082   3.2197514   1.1830635 ]
   [ 2.38314     3.0333805   1.1642089   4.0897436   4.6930127 ]]]]

Relu Activation output:
 [[[[2.0705428  4.8187284  1.7726908  4.627082   0.        ]
   [4.862919   4.722542   0.         4.540087   1.2682633 ]
   [0.         3.1376042  2.9521737  2.066959   0.        ]
   [0.13226175 1.7998643  2.1514082  3.2197514  1.1830635 ]
   [2.38314    3.0333805  1.1642089  4.0897436  4.6930127 ]]]]
-----------------------------------------------------------------------

                        Softmax Activation
Input tensor:
 [[[[ 1.2252657   0.3615985   0.20155334 -0.65938103  4.581312
    -0.3194884 ]]]]

Softmax Activation output:
 [[[[0.03244466 0.01367909 0.01165603 0.00492778 0.9303699  0.00692253]]]]
```

As we know that ReLU sets all negative values to zero and the rest of the values remain the same (as also shown in the figure below), we can understand our output results now.

And for softmax, it takes a vector of real numbers and normalizes the values into a probability distribution.

6.6.1. Dense Layer

We import a dense layer from Keras.layers.core submodule of Keras. The parameters for a dense layer are as follows:

Dense (units, activation=None, use_bias=True)

- Units: Number of neurons in the layer or the number of dimensions in the output.
- Activation: Activation function to be applied to a dense layer output. If this is none, then no activation is applied.
- use_bias: True if we want to use a bias vector.

An input tensor is flattened before feeding into a dense layer if its rank is greater than 2.

PYTHON CODE:

```
# Import Dense and Flatten submodule from keras layers.core
module

from keras.layers.core import Dense, Flatten

# Import keras backend
from keras import backend as K

flatteninput=Flatten()
denselayer=Dense(16)

# A random tensor with lowest value = 0 and highest value =
255

randomtensor=K.random_uniform_
variable(shape=(1,1,5,5),low=0,high=255)

print('Input tensor shape: ',randomtensor.shape,'\n',K.get_
value(randomtensor))

# Flatten the input tensor

outtensor=denselayer(flatteninput(randomtensor))
print('\nOutput shape: ',outtensor.shape)

print('Dense Layer output: \n',K.get_value(outtensor))
```

```
[ ]  # Import Dense and Flatten submodule from keras layers.core module
     from keras.layers.core import Dense, Flatten
     # Import keras backend
     from keras import backend as K

     flatteninput=Flatten()
     denselayer=Dense(16)

     # A random tensor with lowest value = 0 and highest value = 255
     randomtensor=K.random_uniform_variable(shape=(1,1,5,5),low=0,high=255)

     print('Input tensor shape: ',randomtensor.shape,'\n',K.get_value(randomtensor))

     # Flatten the input tensor
     outtensor=denselayer(flatteninput(randomtensor))
     print('\nOutput shape: ',outtensor.shape)

     print('Dense Layer output: \n',K.get_value(outtensor))
```

```
Input tensor shape:  (1, 1, 5, 5)
 [[[[116.745514  46.895596 172.45409  111.540245 183.3954  ]
   [ 96.42691   78.021065 225.5935    60.60836   18.71512 ]
   [ 85.22155   34.107853 170.36714  206.2946   103.72082 ]
   [ 64.65709  102.414566  20.466675 194.40715  159.80948 ]
   [138.83682  249.53848   15.266547  18.376392 138.70067 ]]]]

Output shape:  (1, 16)
Dense Layer output:
 [[ -42.4599     71.85665   142.94595   157.31287    -8.780102 -126.94177
    32.009853  119.2424    119.61725    73.246895 -164.97363   108.78273
  -307.54883   -70.09828   318.86188    -4.682003]]
```

6.7. Model Building

The layers that we have just covered are basic building blocks of a deep learning model. Now, we will stack them or combine them together to build our deep learning models. For now, we will build a simple model to help us understand the concept of deep learning models. Then, we will move to an actual training model for classification tasks.

Here, we will build a model of:

- 1 Convolutional layer
- 1 Activation function
- 1 Maxpool layer

- 1 Dense layer
- 1 Final softmax layer

The Final Dense layer has neurons equal to the number of classes, and the output of the final layer in a model is called logits. In our example, we will have a dense layer of two neurons (because we are building a model to classify between two classes).

PYTHON CODE:

```
# First we import Sequential datastructure of keras

from keras.models import Sequential

# Import Dense, Activation and Flatten submodules from layers.
core module of keras

from keras.layers.core import Dense, Activation, Flatten

# Import Conv2D submodule
from keras.layers.convolutional import Conv2D

# Import MaxPooling2D submodule
from keras.layers import MaxPooling2D

# Import keras backend
from keras import backend as K

# We define our model as an instance of the data structure
model = Sequential()

# Now, we will add layers to the model. First, we add
convolutional layer with no activation and padding = 'valid'

model.add(Conv2D(8, (3,3), strides=(1, 1), padding='valid',
activation=None, use_bias=True,data_format="channels_first"))
```

```
# Add relu activation
model.add(Activation('relu'))

# Add maxpooling layer with padding='valid'

model.add(MaxPooling2D(pool_size=(2, 2), strides=None,
padding='valid', data_format="channels_first"))

# Here, we will have 3 dimensional tensor of some size. For
Dense layer, we will flatten it out.

model.add(Flatten())

# Dense layer with 2 output units
model.add(Dense(2,use_bias=True))

# Softmax activation function
model.add(Activation('softmax'))
```

6.8. Model Summary

Keras provides a method 'summary' to get the summary representation of our model. It lists down the layers of our model, their output shape, and learnable parameters in each layer. To get the summary of a model, we either need to specify the input shape in the first layer of the model or build the model with a specific input shape. For example, the summary of our model is given below:

PYTHON CODE:

```
model.build(input_shape=(1,3,64,64))
model.summary()
```

```
[ ]  model.build(input_shape=(1,3,64,64))
     model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (1, 8, 62, 62)            224
_________________________________________________________________
activation_3 (Activation)    (1, 8, 62, 62)            0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (1, 8, 31, 31)            0
_________________________________________________________________
flatten_2 (Flatten)          (1, 7688)                 0
_________________________________________________________________
dense_2 (Dense)              (1, 2)                    15378
_________________________________________________________________
activation_4 (Activation)    (1, 2)                    0
=================================================================
Total params: 15,602
Trainable params: 15,602
Non-trainable params: 0
_________________________________________________________________
```

- As our convolution layer has 8 kernels, each kernel has a width and a height of 3, and the depth of the kernel is influenced by the number of channels in input tensor. In our case, the images have 3 channels, and so, each one of our kernels has a total of (3 x 3 x 3) dimensions. We need to train each value in each tensor to look for certain pattterns in the image and, therefore, we have total trainable parameters: **(3 * 3 * 3) * 8 + 8 (bias vector) = 224**.
- The output from the flatten_2 layer is 7688 length vector. The dense layer (dense_2) is a fully connected layer and has two neurons. Thus, the total parameters for the dense layer are: **7688 * 2 (7688 connections from the previous layer to each neuron) + 2 (bias units) = 15378.**

6.9. Model Execution

PYTHON CODE:

```
# Now that we have built our model, we would like to see how a
certain input
# tensor will be transformed
# if passed through this model

# Random uniform variable of shape (1 x 3 x 64 x 64) with mean
= 0 and standard deviation = 13

inputtensor=K.random_normal_
variable(shape=(1,3,64,64),mean=0,scale=13)

# Pass this random tensor to model

output=model(inputtensor)
print('output shape: ',output.shape)
print('output values: \n',K.get_value(output)[0])
```

Model Execution

```
[ ]  # Now that we have built our model, we would like to see how a certain input
     # tensor will be transformed
     # if passed through this model

     # Random uniform variable of shape (1 x 3 x 64 x 64) with mean = 0 and
     # standard deviation = 13
     inputtensor=K.random_normal_variable(shape=(1,3,64,64),mean=0,scale=13)
     # Pass this random tensor to model
     output=model(inputtensor)
     print('output shape: ',output.shape)
     print('output values: \n',K.get_value(output)[0])
```

```
output shape:  (1, 2)
output values:
 [2.8046618e-14 1.0000000e+00]
```

As you can see we have output of 2 values, both sum up to 1.0, which results from softmax activation.

6.10. Model Compiling

All the examples that we have covered so far did not include any training. Layers were initialized and were used to process

our input tensors. The weights or values of kernels were pre-initialized. But to solve our problems, we want our kernel windows to look for specific patterns in input data and base their results on that. As we have already discussed how model training works in the theoretical part, we will cover model training steps using Keras here.

After building our model, we need to compile it, which means we need to configure the learning process for our model, which includes the choosing loss function, learning rate, optimizer for weights, updates, etc. Keras provides compile function, and some of the important arguments are mentioned here:

```
compile(optimizer, loss=None, metrics=None, loss_weights=None)
```

- optimizer: It can be a string value defining the name of the optimizer or an instance of the optimizer class. Keras provides a number of optimizers to choose from, e.g., RMSProp, SGD, Adagrad, AdaDelta, Adam, etc.
- loss: It can be a string value defining the name of the loss function to use or can be a defined objective function. Keras provides some loss functions: 'mean_squared_error', 'mean_absolute_error', 'binary_crossentropy', 'categorical_crossentropy', etc.
- metrics: Metric quantifies the fitness of the model to our data, i.e., it measures how well the model fits the data. It can be a list of metrics that the model evaluates during training and testing.
- loss_weights: For multiple classes, we can assign different weights to different classes. It essentially means that a small loss value in one class would be penalized more than the same value loss for any other class. This is particularly important for training on an

imbalanced dataset.

Consider the configurations below where we have used categorical_crossentropy loss, accuracy metric, and rmsprop optimizer for our model.

PYTHON CODE:

```
#Compile the built model (configure the learning process of the
model) by specifying the optimizer used, loss used and metrics

model.compile(loss='binary_crossentropy',
metrics=['accuracy'],optimizer='rmsprop')
```

6.11. Data Building

Now, we have built our model completely. To train this model, we need data. We will use some real-world data later, but for our understanding, we will randomly generate some input tensors and will train our model. In this part, we will focus on learning how to feed data to a model.

We will build a dummy data with 1000 random images of (64 x 64 x 3) dimension each. We have built a binary classifier model, which means that it can classify two classes. So, to use this model, we will somehow label these random images into two classes. Just for learning, we have used the mean of the sum of pixel values to assign classes to each image. Following are the steps:

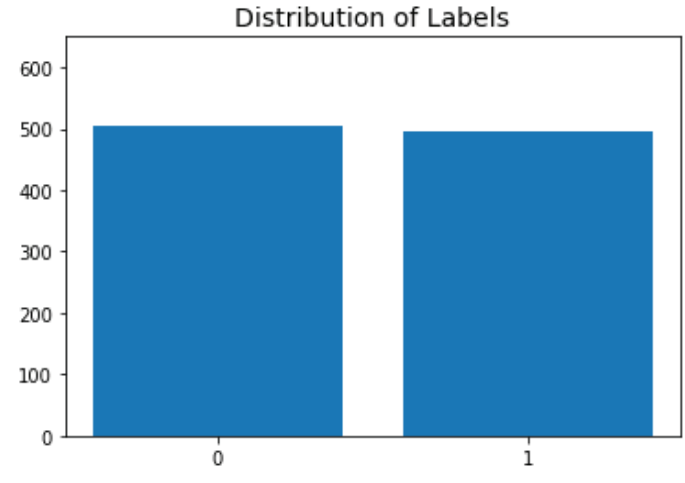
1. The sum of each image along all dimensions (64 x 64 x 3) is calculated.
2. The mean of sums is calculated.
3. All the values greater or equal to mean are assigned class 1 while the rest are assigned class 0.

4. The class numbers are converted into one hot encoding (which is the required format to be used to train the model).

PYTHON CODE:

```
# Import numpy library

import numpy as np

# To preprocess the data. to_categorical is used to get data
in one-hot encoding form

from keras.utils import to_categorical

# Pyplot module of matplotlib
import matplotlib.pyplot as plt

# random uniform sample
totalsamples=1000

X=np.random.
uniform(low=0.0,high=1.0,size=(totalsamples,3,64,64))*255
X=X.astype(int)

# Sum across all values of X across dimension = 1

sums=X.sum(axis=tuple(range(1,X.ndim)))

# Find all values sums
mean=np.mean(sums)

labels=np.zeros(shape=sums.shape,dtype=int)

# Shortlist all the sums greater than mean
labels[sums>=mean]=1

# To show the distribution of our classes

plt.figure('Figure')
plt.title('Distribution of Labels',fontsize=14)
```

```
# Find unique numbers

l, counts = np.unique(labels, return_counts=True)

# Plot a bar of the counts that I have put in

plt.bar(l, counts, align='center')
plt.gca().set_xticks(l)
plt.ylim(0,650)
plt.show()

print('X shape: ',X.shape)
print('Labels shape: ',labels.shape)
```




```
X shape:  (1000, 3, 64, 64)
Labels shape:  (1000,)
```

X shape values are as (batchsize, channels, height, width).

6.12. One Hot Encoding

To train our model, we need to provide our input images and their labels in the correct format. Our images have two classes (0 and 1), and so, our labels are integer encoded (numbered 0 and 1). One hot encoding is used to represent our categorical variables as binary vectors (in the form of 0s and 1s). The length of the vector is equal to the number of classes. For a label integer, the binary vector has a 1 at the index of value and 0s at all other positions. To better understand this, consider the following example:

We have 5 classes: {0,1,2,3,4}. We can represent each of those values as a binary vector of length 5. Following are their binary vectors:

- 0: [1,0,0,0,0]
- 1: [0,1,0,0,0]
- 2: [0,0,1,0,0]
- 3: [0,0,0,1,0]
- 4: [0,0,0,0,1]

As you can see, all the values in a vector are 0, with 1 at the index position of that value.

We can write custom code using numpy to get one hot encoded vectors, or we can use the Keras library to do that.

PYTHON CODE:

```
# In this cell, we will build one hot encoded vectors for
categorical classes
#Import numpy library

import numpy as np

# Total 5 classes

classes=[0,1,2,3,4]
for c in classes:

    # A one hot encoded form of classes

    one_hot=np.zeros(shape=(len(classes)))
    one_hot[c]=1
    print('Class',c,': ',one_hot)
print('\n')

# Now, let's take another example of categorical variables

classes=['red','green','blue','orange','yellow','purple','indigo']

for idx,c in enumerate(classes):
    one_hot=np.zeros(shape=(len(classes)))
    one_hot[idx]=1
    print('%5s %7s %s %15s '%('Class',c,':',one_hot) )

    #print('Class',c,': ',one_hot)
```

```
Class 0 :  [1. 0. 0. 0. 0.]
Class 1 :  [0. 1. 0. 0. 0.]
Class 2 :  [0. 0. 1. 0. 0.]
Class 3 :  [0. 0. 0. 1. 0.]
Class 4 :  [0. 0. 0. 0. 1.]


Class     red : [1. 0. 0. 0. 0. 0. 0.]
Class   green : [0. 1. 0. 0. 0. 0. 0.]
Class    blue : [0. 0. 1. 0. 0. 0. 0.]
Class  orange : [0. 0. 0. 1. 0. 0. 0.]
Class  yellow : [0. 0. 0. 0. 1. 0. 0.]
Class  purple : [0. 0. 0. 0. 0. 1. 0.]
Class  indigo : [0. 0. 0. 0. 0. 0. 1.]
```

PYTHON CODE:

```
# Keras provides one line solution for this. So, we will use
to_categorical to get our labels converted to one-hot encoding

one_hot_labels=to_categorical(labels)
```

Now we have 1000 input tensors and their labels to train our model. Before that, we will split our data into train, validation, and test data. So, we can validate our results on the go and later test it. First, we will use the sklearn's train_test_split function to split our data into train and test parts. Then, we will further divide the train set into train and validation set.

PYTHON CODE:

```
from sklearn.model_selection import train_test_split

# Following function will randomly shuffle our data and will
split it into 80 - 20 percents(train and test data)

X_train,X_test,y_train,y_test=train_test_split(X,one_hot_
labels,test_size=0.2,random_state=17)

# Following function will divide train set into 90 - 10 split
of train and validation set.

X_train,X_val,y_train,y_val=train_test_split(X_train,y_
train,test_size=0.1,random_state=43)

print('X_train shape: ',X_train.shape)
print('y_train shape: ',y_train.shape)

print('X_val shape  : ',X_val.shape)
print('y_val shape  : ',y_val.shape)

print('X_test shape : ',X_test.shape)
print('y_test shape : ',y_test.shape)
```

```
from sklearn.model_selection import train_test_split

# Following function will randomly shuffle our data and will split it into 80 - 20 percents(train and test data)
X_train,X_test,y_train,y_test=train_test_split(X,one_hot_labels,test_size=0.2,random_state=17)

# Following function will divide train set into 90 - 10 split of train and validation set.
X_train,X_val,y_train,y_val=train_test_split(X_train,y_train,test_size=0.1,random_state=43)

print('X_train shape: ',X_train.shape)
print('y_train shape: ',y_train.shape)

print('X_val shape  : ',X_val.shape)
print('y_val shape  : ',y_val.shape)

print('X_test shape : ',X_test.shape)
print('y_test shape : ',y_test.shape)
```

```
X_train shape:  (720, 3, 64, 64)
y_train shape:  (720, 2)
X_val shape  :  (80, 3, 64, 64)
y_val shape  :  (80, 2)
X_test shape :  (200, 3, 64, 64)
y_test shape :  (200, 2)
```

6.13. Model Training

To train a model, we call the "fit" function of the model.

```
fit(x=None, y=None, batch_size=None, epochs=1, verbose=1,
callbacks=None, validation_split=0.0, validation_data=None,
shuffle=True, class_weight=None, sample_weight=None, initial_
epoch=0, steps_per_epoch=None, validation_steps=None,
validation_freq=1)
```

- x: It is a numpy array of input training data, e.g., images. In our case, it will contain our input tensors.
- y: It is also a numpy array of training data labels. In our case, it will contain one hot encoded vectors.
- batch_size: Number of samples provided to a model for each run and for gradient calculation.
- epochs: Number of epochs represents the number of times a model will iterate through the complete dataset. For example, epochs = 10 means that the model will iterate through the dataset 10 times.
- verbose: If verbose = 1, in-between epochs results will be printed.
- callbacks: Function callbacks during training or validation. This is a list of functions that will be called during training or validation.
- validation_split: This is a value between 0 and 1. It represents the percentage of training data to be used for validation.
- validation_data: It is a tuple containing validation input data and labels (Xval, Yval). The model will be evaluated on this data after each epoch.
- shuffle: It is a boolean value. If shuffle = True, the data will be randomly shuffled before each iteration/epoch.

- initial_epoch: Epoch number from which to start the training. It is useful to resume a previous training.

This fit function returns a history object that contains records of training and validation losses and metrics values at successive epochs. Through this history object, we can visualize the losses and accuracies by plots.

PYTHON CODE:

```
history= model.fit(x=X_train,y=y_train,batch_
size=1,epochs=2,verbose=1,validation_data=(X_val,y_
val),shuffle=True)
```

We trained the model for two epochs. Our validation results remained the same while our training accuracy increased, and our training loss decreased a little.

PYTHON CODE:

```
# This function will take history/output from train/test
function.

def summarize_stats(history,title):

    # loss
    # Set figure size

    plt.figure(figsize=(16,4))

    # Add subplots

    plt.subplot(1,2,1)

    # Add suptitle

    plt.suptitle(title,fontsize=18)

    # Get a figure for loss
    plt.title('Loss',fontsize=16)
```

```
    # Plot history of the training/testing
    plt.plot(history.history['loss'], color='red',
label='Train')
    plt.plot(history.history['val_loss'], color='blue',
label='Test')
    plt.legend(loc='upper right')

    # accuracy
    plt.subplot(1,2,2)

    # Figure of classification accuracy
    plt.title('Classification Accuracy',fontsize=16)

    # Plot accuracy from history plot
    plt.plot(history.history['acc'], color='red',
label='Train')

    # Plot validation accuracy
    plt.plot(history.history['val_acc'], color='blue',
label='Test')

    plt.legend(loc='lower right')
    plt.show()


summarize_stats(history,'Reuslts')
```

6.14. Pretrained Models and Fine-Tuning

A deep learning model requires a huge amount of data to train well. Usually, for our particular problem, we only have access to a small dataset. If we train our model from scratch with randomly initialized weights, the model will most likely overfit the training set. To tackle this problem, we use models that have been pretrained on some large dataset, and then we initialize our layers with weights of those models. This process is called fine-tuning.

In fine-tuning, we tune the network parameters/weights of an already trained model so it can adapt to the new task. Initial layers in a model learn the general features while the last layers learn the features that are specific to our task, e.g., initial layers might learn circular edges, horizontal or vertical edges, while last layers learn specific features like the shape of the nose, lips, ear, etc. Therefore, in fine-tuning, we only train the last few layers and freeze the initial layers. To fine-tune a model, we replace the last few layers to get our required number of classes as output, and then we train the complete model.

Keras provides models pretrained on ImageNet, which we can use for our tasks. Some of those models are VGG16, VGG19, InceptionV3, ResNet, and DenseNet.

Consider the example of using VGG16 provided by Keras. We can import the models from the Keras.applications module. To initialize a model, we call the module with parameters:

```
VGG16(include_top=True, weights='imagenet', input_shape=None,
pooling=None, classes=1000)
```

- include_top: If True, the last fully connected layers will be included, and if False, the last fully connected layers will be excluded.
- weights: If None, then the model weights will be randomly initialized, and if 'imagenet', then the weights of a model trained on imagenet will be loaded.
- input_shape: To specify the shape of the input tensor. It has to be specified only if include_top is False.
- pooling: To be specified when include_top is False. This is for feature extraction. Possible values are: None for no pooling, 'avg' for average pooling, and 'max' for max pooling applied to the output of the last conv layer.

- classes: Number of classes to classify images into. This should be specified only if include_top is True and weights is None.

PYTHON CODE:

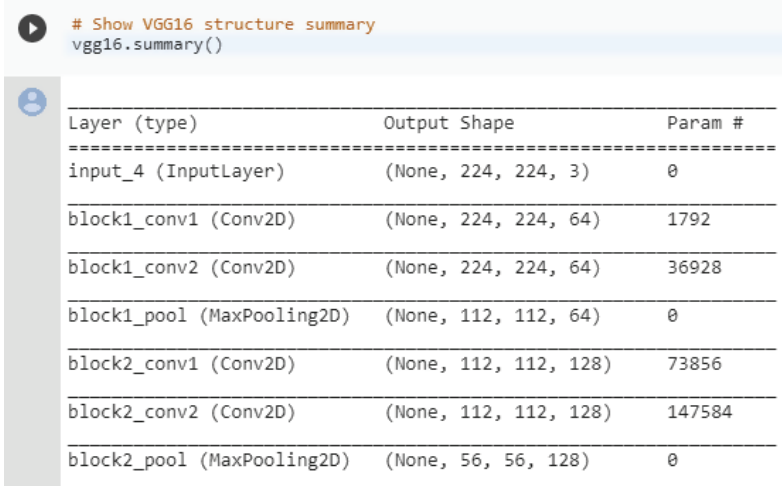
```
# Import pretrained prebuilt VGG16 model architecture

from keras.applications import VGG16

# Specifying the input shape and using weights of 'imagenet'
trained model

vgg16=VGG16(weights='imagenet',input_
shape=(224,224,3),include_top=True)
```

6.15. Model Summary

As we included the top layers using include_top=True, we can look at the complete model:

PYTHON CODE:

```
# Show VGG16 structure summary

vgg16.summary()
```



```
# Show VGG16 structure summary
vgg16.summary()

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
```

```
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
predictions (Dense)          (None, 1000)              4097000
=================================================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
_________________________________________________________________
```

6.15.1. Steps for Fine Tuning:

To fine-tune this model, we need to remove the last fully connected and flatten layers, and add the layers specific to solve our problem. For that, we need to follow these steps:

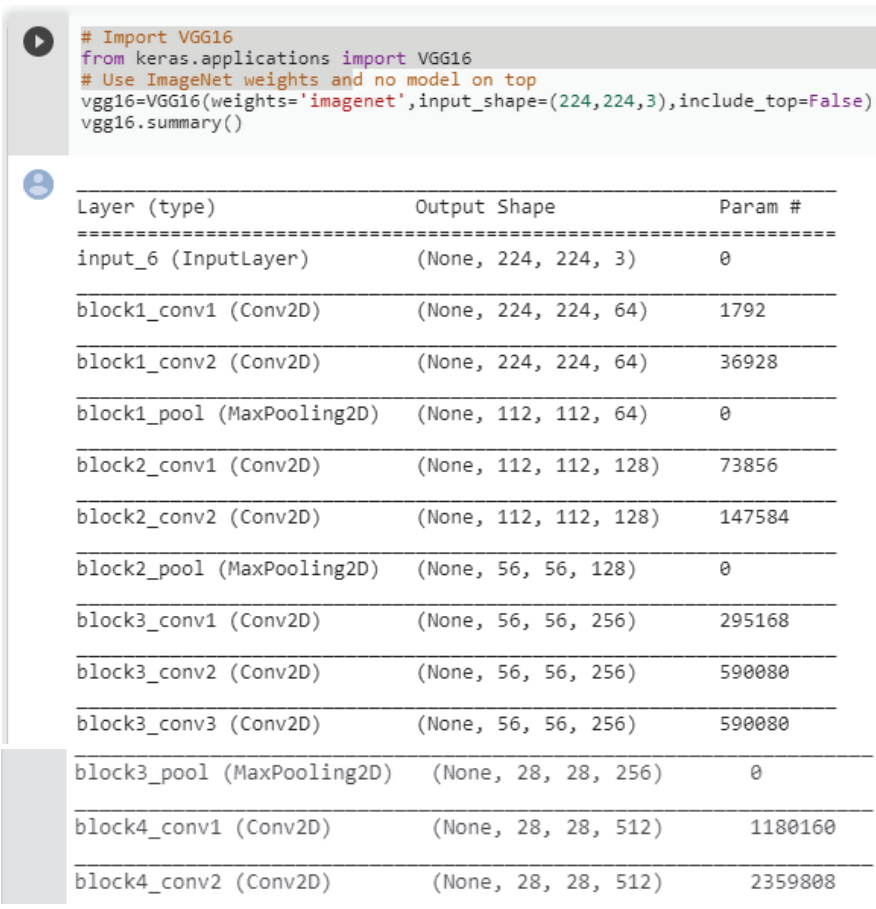
1. Load pretrained model and set include_top=False to remove last fully connected layers.
2. Freeze initial layers because we do not want to train them.
3. Build our own model on top of the pretrained one.
4. Train the model.

Step 1: Load Pretrained Model

We will load a pretrained VGG16 with include_top=False.

PYTHON CODE:

```
# Import VGG16
from keras.applications import VGG16

# Use ImageNet weights and no model on top

vgg16=VGG16(weights='imagenet',input_
shape=(224,224,3),include_top=False)

vgg16.summary()
```



```
# Import VGG16
from keras.applications import VGG16
# Use ImageNet weights and no model on top
vgg16=VGG16(weights='imagenet',input_shape=(224,224,3),include_top=False)
vgg16.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_6 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
```

```
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

As you can see, the last FC layers have been removed, and our last layer is a block5_pool layer.

Step 2: Freeze Initial Layers

As we can see from the model summary, it has five blocks, and each block has Conv and Pool layers. We do not want to train the initial layers, so we will freeze all the layers in the first four blocks and will train the last four layers. Keras layers have a property 'trainable'. We can set it to False if we do not want to train a layer:

PYTHON CODE:

```
# Freeze the last 4 layers and train them accordingly

for layer in vgg16.layers[:-4]:
    layer.trainable=False

#Checking layers which are trainable:

for layer in vgg16.layers:
    print(layer,layer.trainable)
```

```
[ ]  # Freeze the last 4 layers and train them accordingly
     for layer in vgg16.layers[:-4]:
         layer.trainable=False

     #Checking layers which are trainable:
     for layer in vgg16.layers:
         print(layer,layer.trainable)
```

```
<keras.engine.input_layer.InputLayer object at 0x7fb33bd95470> False
<keras.layers.convolutional.Conv2D object at 0x7fb33bdad860> False
<keras.layers.convolutional.Conv2D object at 0x7fb33bdada58> False
<keras.layers.pooling.MaxPooling2D object at 0x7fb33bd49f28> False
<keras.layers.convolutional.Conv2D object at 0x7fb33bd49be0> False
<keras.layers.convolutional.Conv2D object at 0x7fb33bd7bac8> False
<keras.layers.pooling.MaxPooling2D object at 0x7fb33bd13d68> False
<keras.layers.convolutional.Conv2D object at 0x7fb33bd13e48> False
<keras.layers.convolutional.Conv2D object at 0x7fb33bccbe10> False
<keras.layers.convolutional.Conv2D object at 0x7fb33bcfa668> False
<keras.layers.pooling.MaxPooling2D object at 0x7fb33bc96dd8> False
<keras.layers.convolutional.Conv2D object at 0x7fb33bc96a90> False
<keras.layers.convolutional.Conv2D object at 0x7fb33bc48978> False
<keras.layers.convolutional.Conv2D object at 0x7fb33bc65f98> False
<keras.layers.pooling.MaxPooling2D object at 0x7fb33bc199e8> False
<keras.layers.convolutional.Conv2D object at 0x7fb33bc19518> True
<keras.layers.convolutional.Conv2D object at 0x7fb33bbcd240> True
<keras.layers.convolutional.Conv2D object at 0x7fb33bbe4c50> True
<keras.layers.pooling.MaxPooling2D object at 0x7fb33bb9a588> True
```

Step 3: Build a Model on Top of Pretrained

Since our pretrained model already has Conv and Pool layers, we can focus on adding fully connected layers only and add them on top of the pretrained model. This approach provides us the flexibility to mold a model pretrained for some problem to our solve our own specific problem, e.g., we can add layers for other problems like regression, image localization, and image segmentation.

PYTHON CODE:

```
# Add sequential module from models in keras library

from keras.models import Sequential

# Import Dense, Activation and Flatten submodules of layers.
core module

from keras.layers.core import Dense, Activation, Flatten

# Import convolutional layer from keras.layers

from keras.layers.convolutional import Conv2D

# Import maxpooling layer from keras.layers

from keras.layers import MaxPooling2D

# Import dropout layer from keras

from keras.layers import Dropout

def build_model_finetune(basemodel):

    # A sequential array which can contain the model layers

    model=Sequential()

    # Add base model
    # Add basic VGG16 initially

    model.add(basemodel)

    # Flatten the image pixels
    # Flatten out the output from VGG16

    model.add(Flatten())
```

```
    # Add fully connected layers
    # A dense layer with 512 output units
    model.add(Dense(512))

    # A relu activation function
    model.add(Activation('relu'))

    # A dropout of 0.2
    model.add(Dropout(0.2))

    # A fully connected layer for 2 classes
    model.add(Dense(2))

    #a softmax to get probabilities of each class
    model.add(Activation('softmax'))

    return model

model = build_model_finetune(vgg16)
model.compile(loss='categorical_crossentropy',
metrics=['accuracy'],optimizer='rmsprop')
model.summary()
```

```
    return model

model = build_model_finetune(vgg16)
model.compile(loss='categorical_crossentropy', metrics=['accuracy'],optimizer='rmsprop')
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 7, 7, 512)         14714688
_________________________________________________________________
flatten_3 (Flatten)          (None, 25088)             0
_________________________________________________________________
dense_4 (Dense)              (None, 512)               12845568
_________________________________________________________________
activation_4 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_5 (Dense)              (None, 2)                 1026
_________________________________________________________________
activation_5 (Activation)    (None, 2)                 0
=================================================================
Total params: 27,561,282
Trainable params: 19,926,018
Non-trainable params: 7,635,264
_________________________________________________________________
```

As you can see, the model now has vgg16 as the base model, and our defined layers are on top of the base model for classifying two classes.

Step 4: Train the Model

Finally, we train the model using .fit() function provided by Keras, as explained above.

7

Project 1: Linear Regression with One Variable

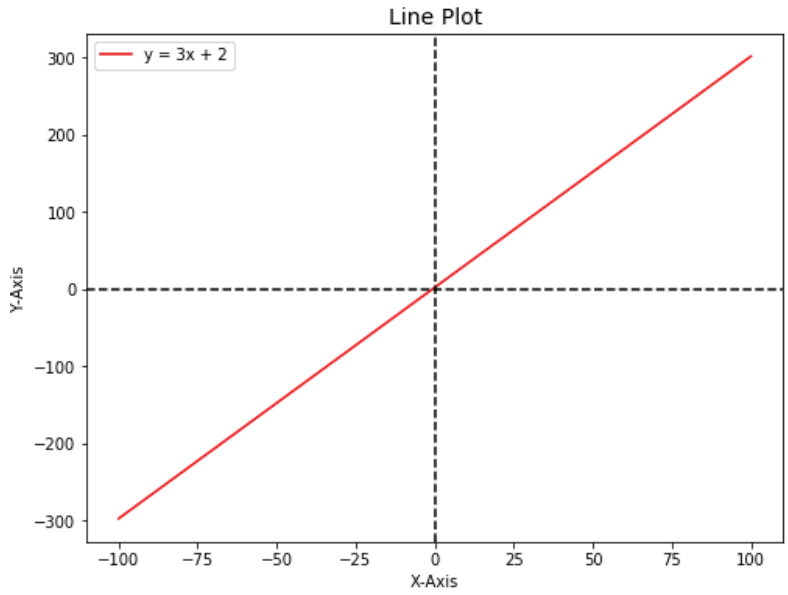
This chapter covers:

1. Generating linear data
2. Using Ordinary Least Squares Method to estimate parameters
3. Using Normal Equation to estimate parameters
4. Building Regression Model and training it with gradient descent
5. Comparing results for estimated parameters from different methods.

Consider a line: y = 3x + 2. Let's draw this between x = -100 and x = 100

PYTHON CODE:

```
# Import pyplot library from matplotlib
import matplotlib.pyplot as plt

# Import numpy library
import numpy as np

# Get 1000 equally distant values between -100 and +100

x=np.linspace(-100,100,1000)

# The equation of line:

y=3*x+2

# We need one long vector for each x and y
# Reshape arrays to have 1 channel in empty dimension

x=np.reshape(x,(-1,1))
y=np.reshape(y,(-1,1))

# Create a figure to plot
plt.figure(figsize=(8,6))

# Set title of the figure
plt.title('Line Plot',fontsize=14)

# Plot x and y points as a line
plt.plot(x,y,c='red',label='y = 3x + 2')

# Draw X and Y Axis
plt.axhline(y=0,c='black',linestyle='--')
plt.axvline(x=0,c='black',linestyle='--')

# Set X and Y axis labels
plt.xlabel('X-Axis')
plt.ylabel('Y-Axis')

# Add legend on upper left side
plt.legend(loc='upper left')

plt.show()
```

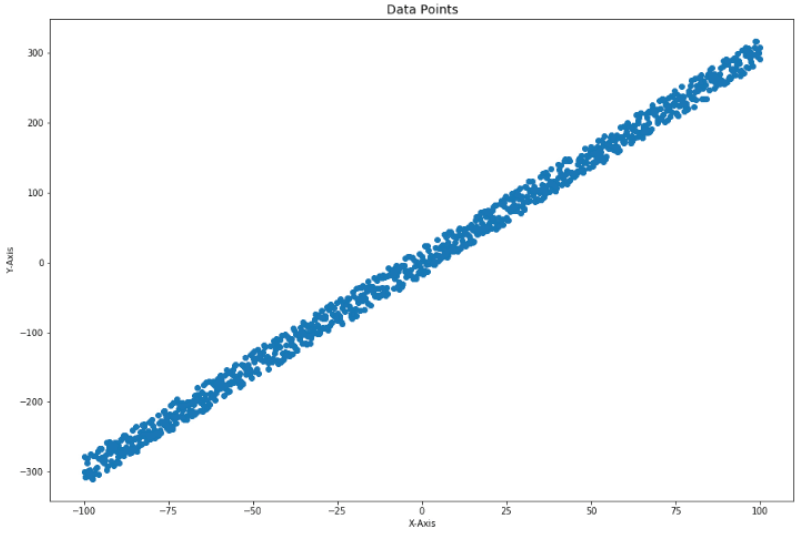


This line is actually a set of points that lie on a straight line. We want to build a regression model that, given a set of points (x,y), can accurately map x to y. In other words, it can estimate the parameters of the equation of line: y = mx + c (m = 3, c = 2) in our case. Rather than having a linear relationship between x and y, we will add noise to our output values (y). Thus, we won't be able to fit any linear model perfectly. We will try to find the best model line that can fit those data points.

First, we will generate a vector of noise (random values) of the same shape as of y vector, and then, we will add them together. First, we will generate random numbers between −1 and 1 drawn from uniform distribution. Then, we multiply those floating values to a fixed number (offset). This will expand all the values by a fixed factor.

PYTHON CODE:

```
# Add some noise to y values, so we don't get a perfect fit of
the line

offset=20

#Add some offset of randomly generated numbers
noise=np.random.uniform(-1,1,size=y.shape)*offset

# Adding noise to Y
y = y + noise

# Draw (x,y) points
plt.figure(figsize=(15,10))
plt.title('Data Points',fontsize=14)
plt.xlabel('X-Axis')
plt.ylabel('Y-Axis')

# Plot points (x,y) pairs

plt.scatter(x,y)
plt.show()
```



Using the sklearn library, we will divide our data input train and test sets.

PYTHON CODE:

```
# Importing from sklearn, to split dataset into train and test
split

from sklearn.model_selection import train_test_split

# Given x and y list values, this function will split the data
into train and test data by randomly selecting points for each
(train and test)

train_x,test_x,train_y,test_y=train_test_split(x,y)
```

7.1. Estimating Parameters:

Equation of a line is given by y = mx + c where m and c are parameters. We need to estimate those parameters. For that, we will use three methods and will compare their results.

7.1.1. Ordinary Least Squares Method

The OLS method is given by the algorithm:

Algorithm 1 Linear Regression

$$\bar{x} \leftarrow \frac{\sum x^{(i)}}{m} \ , \ \bar{y} \leftarrow \frac{\sum y^{(i)}}{m}$$

$$SP_{xy} \leftarrow \sum_{i=0}^{m-1} (x^{(i)} - \bar{x})(y^{(i)} - \bar{y})$$

$$SP_{xx} \leftarrow \sum_{i=0}^{m-1} (x^{(i)} - \bar{x})^2$$

$$c_1 \leftarrow \frac{SP_{xy}}{SPxx}$$

$$c_0 = \bar{y} - c_1 \bar{x}$$

PYTHON CODE:

```
# Following lines implement the algorithm mentioned in the
image above.
# Calculate mean values for x array and y array values

xmean=np.mean(x)
ymean=np.mean(y)

SP_xy=np.sum((x-xmean)*(y-ymean))
SP_xx=np.sum((x-xmean)**2)
OLS_c_1=SP_xy/SP_xx
OLS_c_0=ymean-OLS_c_1*(xmean)

print('Calculated parameter values using OLS method are: %.3f
%.3f'%(OLS_c_0,OLS_c_1))
```

```
Calculated parameter values using the OLS method are: 1.545
3.006
```

7.1.2. Normal Equation

Now, we will use the normal equation method to estimate our parameters.

$$C = (X^T X)^{-1} X^T Y$$

$$X = \begin{bmatrix} 1 & x^{(0)} \\ 1 & x^{(1)} \\ \vdots & \vdots \\ 1 & x^{(m)} \end{bmatrix}, Y = \begin{bmatrix} y^{(0)} \\ y^{(1)} \\ \vdots \\ (m) \end{bmatrix}$$

Here, we will create a matrix of X by appending the 1s column with our original vector x. Then, we will use this equation to calculate our parameters directly.

PYTHON CODE:

```
# Initializing an array of shape as x array, filled with ones

ones=np.ones(shape=x.shape)

# Appending ones as column with x
X=np.concatenate((ones,x),axis=1)
Y=y

# Equation to get coefficients

[Normal_c_0,Normal_c_1]=np.dot(np.dot(np.linalg.pinv(np.
dot(X.T,X)),X.T),Y)

print('Calculated parameter values using Normal equation are:
%.3f %.3f'%(Normal_c_0,Normal_c_1))
```

```
Calculated parameter values using Normal equation are: 1.545
3.006
```

7.2. Using Gradient Descent

We will use the gradient descent method to estimate our parameters. The method of gradient descent has been explained well in the theoretical part. So the steps to perform a single iteration of weight updates is as follows:

1. We start by initializing our parameter values. Here, we have initialized them randomly.
2. Convert x, y, and weights to matrices (numpy arrays).
3. Compute cost.
4. Compute the derivative of the cost.
5. Update parameters by a factor of derivative (factor means the amount of updates to be made in each iteration and is defined by the learning rate).
6. Repeat Point 2 – Point 6 for N number of epochs.

PYTHON CODE:

```
# Initializing parameters randomly

[c_0,c_1]=np.random.rand(2)
```

7.2.1. Compute Derivatives

Derivatives with respect to each parameter will be calculated, and the equations to find them are given by:

$$\frac{\partial J}{\partial c_0} = \frac{\partial J}{\partial c_0}(\frac{1}{m}\sum_{i=0}^{m-1}((c_0x_0^{(i)} + c_1x_1^{(i)}) - y^{(i)})^2)$$

$$= \frac{2}{m}\sum_{i=0}^{m-1}((c_0x_0^{(i)} + c_1x_1^{(i)}) - y^{(i)})x_0^{(i)}$$

$$\frac{\partial J}{\partial c_1} = \frac{\partial J}{\partial c_1}(\frac{1}{m}\sum_{i=0}^{m-1}((c_0x_0^{(i)} + c_1x_1^{(i)}) - y^{(i)})^2)$$

$$= \frac{2}{m}\sum_{i=0}^{m-1}((c_0x_0^{(i)} + c_1x_1^{(i)}) - y^{(i)})x_1^{(i)}$$

PYTHON CODE:

```
# This function computes derivative of cost with respect to a
parameter (c_0 or c_1) where idx is the index of parameter to
which we need to find the derivative with respect to.

def compute_derivative(c_0,c_1,X,Y,m,idx):

  # Compute y values with the parameters of model (c_0 and c_1):

  y_pred=np.reshape(c_0*X[:,0]+c_1*X[:,1],(m,1))

  # Computer derivative of loss function with respect to each
parameter (we have already covered this in theoretical part)

  deriv=np.sum(np.multiply((y_pred-Y),np.
reshape(X[:,idx],(m,1))))

  deriv=(deriv*2)/m
  return deriv
```

7.2.2. Compute Cost

To calculate the cost, we predict y values $\boldsymbol{p}^{(i)}$ using new parameters and use actual y values:

$$J = \frac{1}{m}\sum_{i=0}^{m-1}(p^{(i)} - y^{(i)})^2$$

PYTHON CODE:

```
# This function computes cost/loss of a set of parameters
def compute_cost(c_0,c_1,X,Y,m):
  # Compute y values with parameters of model (c_0 and c_1)

  y_pred=np.reshape(c_0*X[:,0]+c_1*X[:,1],(m,1))

  # Compute cost with formula we covered earlier

  out=np.square(y_pred-Y)
  return (np.sum(out)*2)/m

import math

# We will use r2_score to define the accuracy of regression
model

from sklearn.metrics import r2_score

# learning ratel

r=1e-7

# m = Total number of points for training

m=train_y.shape[0]

# Train the model for 20000 epochs

epochs=20000

Y=train_y
```

```
# Create X array by appending 1s with original x values

X=np.concatenate((np.ones_like(train_x),train_x),axis=1)

losses=[]
train_acc=[]
test_acc=[]

# Compute cost with new paramters.

for i in range(1,epochs):
    loss=compute_cost(c_0,c_1,X,Y,m)
    # If loss is not a number or is infinity, break the loop

    if(math.isnan(loss) or math.isinf(loss)):
        print('breaking at iteration:',i)
        break
    losses.append(loss)
    deriv_0=compute_derivative(c_0,c_1,X,Y,m,0)

    # Compute derivative of the cost with respect of second parameter

    deriv_1=compute_derivative(c_0,c_1,X,Y,m,1)

    # Update parameters by a factor of derivative

    c_0=c_0-lr*deriv_0
    c_1=c_1-lr*deriv_1

    # evaluate training model on train data

    _,r2score=evaluate(train_x,train_y,c_0,c_1)
     train_acc.append(r2score)

    # evaluate training model on test data

    _,r2score=evaluate(test_x,test_y,c_0,c_1)
    test_acc.append(r2score)

GD_c_0=c_0
GD_c_1=c_1

print('Calculated parameter values using Gradient Descent are:
%.3f %.3f'%(GD_c_0,GD_c_1))
```

7.3. Visualizations of Loss and Accuracy Metrics

We will visualize the loss values and accuracy metrics with respect to epochs to see how the model improved with time.

PYTHON CODE:

```
# Define a figure for loss visualization

plt.figure('Loss')

plt.title('Gradient Descent',fontsize=10)

plt.autoscale(enable=True, axis='y')

plt.plot(np.array(range(len(losses))),losses,'-
','color='green',label='loss')

plt.legend(loc='upper right')
plt.xlabel('Epochs',fontsize=8)
plt.ylabel('Loss',fontsize=8)
plt.show()

#we can also visualize that model has not just overfit the
training data but is also performing well on test data

plt.figure('Accuracy Metrics')

plt.title('Train and Test R2 scores in Gradient
Descent',fontsize=10)

plt.autoscale(enable=True, axis='y')

plt.plot(np.array(range(len(train_acc))),train_acc,'-
','color='green',label='Training Evaluation')

plt.plot(np.array(range(len(test_acc))),test_acc,'-
','color='red',label='Test Evaluation')

plt.legend(loc='upper left')
plt.xlabel('Epochs',fontsize=8)
plt.ylabel('r2_score',fontsize=8)

plt.show()
```

7.4. Evaluation

We need to evaluate three models (OLS, Normal Equation, and Gradient Descent) to see which one performed the best. We can use Mean Squared Error Metrics, Mean Absolute Error, Root Mean Squared Error, or R Squared.

We will use R2_score to evaluate our models. An r2 score can range from −1 to 1, with the best value being 1. A model performing the worst will give an r2 score of −1. R2 score compares the model with a horizontal line that passes through the data. This means that a model which always predicts a constant value irrespective of the input value will have an R2 score of 0.0

PYTHON CODE:

```
# Following function will evalute the parameters estimated by
a model

def evaluate(test_x,test_y,c0__,c1__):

  # Get y values based on model parameters
  pred_y=c0__+c1__*test_x

  # Calculate r2_score
  r2score=r2_score(test_y,pred_y)

  # calculate adjusted r2_score

  adjusted_r_squared = 1 - (1-r2score)*(len(test_y)-1)/
(len(test_y)-X.shape[1]-1)

  return pred_y,adjusted_r_squared

# Following function will evalute the parameters estimated by
a model

def evaluate(test_x,test_y,c0__,c1__):
```

```
  # Get y values based on model parameters
  pred_y=c0__+c1__*test_x

  # Calculate r2_score
  r2score=r2_score(test_y,pred_y)

  # calculate adjusted r2_score

  adjusted_r_squared = 1 - (1-r2score)*(len(test_y)-1)/
(len(test_y)-X.shape[1]-1)

  return pred_y,adjusted_r_squared

# These x and y limits are just to define the viewing area in
2D coordinates

xminlim=-50
xmaxlim=50
yminlim=-200
ymaxlim=200

# Define three subplots
fig, ax = plt.subplots(1, 3)

# Set x and y limits for all three subplots

for i in range(3):
    ax[i].set_xlim([xminlim,xmaxlim])
    ax[i].set_ylim([yminlim,ymaxlim])

# Set figure size (in inches)
fig.set_size_inches(10.25, 5.25)

fig.suptitle('Linear Regression',fontsize=12)
```

```
# Plot (x,y) data points on all three subplots

ax[0].scatter(x,y,c='r',marker='.')
ax[1].scatter(x,y,c='r',marker='.')
ax[2].scatter(x,y,c='r',marker='.')

print('\nMethod: Ordinary Least Squares')
# Calculate R2 score for Ordinary Least Squares method for
test and train

_,r2score=evaluate(train_x,train_y,OLS_c_0,OLS_c_1)

print('Training R2_score: ',r2score)
pred_ols,r2score=evaluate(test_x,test_y,OLS_c_0,OLS_c_1)
print('Test R2_score: ',r2score)

ax[0].set_title('Method: Ordinary Least Squares')
p=ax[0].plot(test_x,pred_ols,'-',c='black',label='Method 1')

ax[0].set_xlabel('X Values')
ax[0].set_ylabel('Y Values')

print('\nMethod: Normal Equation')

# Calculate R2 score for Normal Equation method for test and
train

_,r2score=evaluate(train_x,train_y,Normal_c_0,Normal_c_1)
print('Training R2_score: ',r2score)

pred_
normal,r2score=evaluate(test_x,test_y,Normal_c_0,Normal_c_1)

print('Test R2_score: ',r2score)

ax[1].set_title('Method: Normal Equation')
ax[1].plot(test_x,pred_normal,'-',c='blue')
ax[1].set_xlabel('X Values')
ax[1].set_ylabel('Y Values')

print('\nMethod: Gradient Descent')
```

```
# Calculate R2 score for Gradient Descent method for test and
train

_,r2score=evaluate(train_x,train_y,GD_c_0,GD_c_1)
print('Training R2_score: ',r2score)

pred_gd,r2score=evaluate(test_x,test_y,GD_c_0,GD_c_1)

print('Test R2_score: ',r2score)

ax[2].set_title('Method: Gradient Descent')
ax[2].plot(test_x,pred_gd,'-',c='green')
ax[2].set_xlabel('X Values')
ax[2].set_ylabel('Y Values')

plt.show()
```

All three models performed equally well on the dataset.

8

Project 2: Multivariate Regression

This chapter explains how to build a regression model that learns from multiple features of real-world data. Specifically, we will use Boston Housing data to estimate the price of a house based on multiple features. Since our model will predict house prices based on several features (multiple parameters), we call our model *multivariate regression model.*

8.1. Dataset Preparation

We can directly load the data from the sklearn.datasets module. The "load_boston()" function will return a dictionary which will contain our dataset input features, target variables, and some metadata that we can access through ".DESCR" property.

PYTHON CODE:

```
# Import dataset module from sklearn through which, we can
load datasets

from sklearn import datasets

# Load boston dataset
dataset=datasets.load_boston()
```

```
#dataset shape: (506,13) => (Number of data records,Number of
features)

print('Dataset X shape:',dataset.data.shape)
print('Dataset Y shape:',dataset.target.shape)

print(dataset.DESCR)
```

```
Dataset X shape: (506, 13)
Dataset Y shape: (506,)
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

:Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 14) is usually the target.

:Attribute Information (in order):
    - CRIM     per capita crime rate by town
    - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
    - INDUS    proportion of non-retail business acres per town
    - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
    - NOX      nitric oxides concentration (parts per 10 million)
    - RM       average number of rooms per dwelling
    - AGE      proportion of owner-occupied units built prior to 1940
    - DIS      weighted distances to five Boston employment centres
    - RAD      index of accessibility to radial highways
    - TAX      full-value property-tax rate per $10,000
    - PTRATIO  pupil-teacher ratio by town
    - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
    - LSTAT    % lower status of the population
    - MEDV     Median value of owner-occupied homes in $1000's

:Missing Attribute Values: None

:Creator: Harrison, D. and Rubinfeld, D.L.
```

```
This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/

This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics
...', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that address regression
problems.

.. topic:: References

- Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying
  Influential Data and Sources of Collinearity', Wiley, 1980.
  244-261.
- Quinlan, R. (1993). Combining Instance-Based and Model-
  Based Learning. In Proceedings of the Tenth International
  Conference of Machine Learning, 236-243, University of
  Massachusetts, Amherst. Morgan Kaufmann.
```

In Linear Regression with One Variable (chapter 18), we built a linear model with two parameters (m and c), and we estimated those parameters through gradient descent and other methods. We can use a similar approach for this model:

$$\begin{aligned} p^{(i)} &= c_0 x_0^{(i)} + c_1 x_1^{(i)} + c_2 x_2^{(i)} + \cdots + c_n x_n^{(i)} \\ &= \sum_{j=0}^{n} c_j x_j^{(i)} \end{aligned}$$

But with this model, we will only be able to map the linear relationship between independent and dependent variables while our parameters might have a nonlinear relationship with our target variable. Consider the following graphs which plot the correlations between dependent and each independent variable pairs.

8.1.1. Correlation

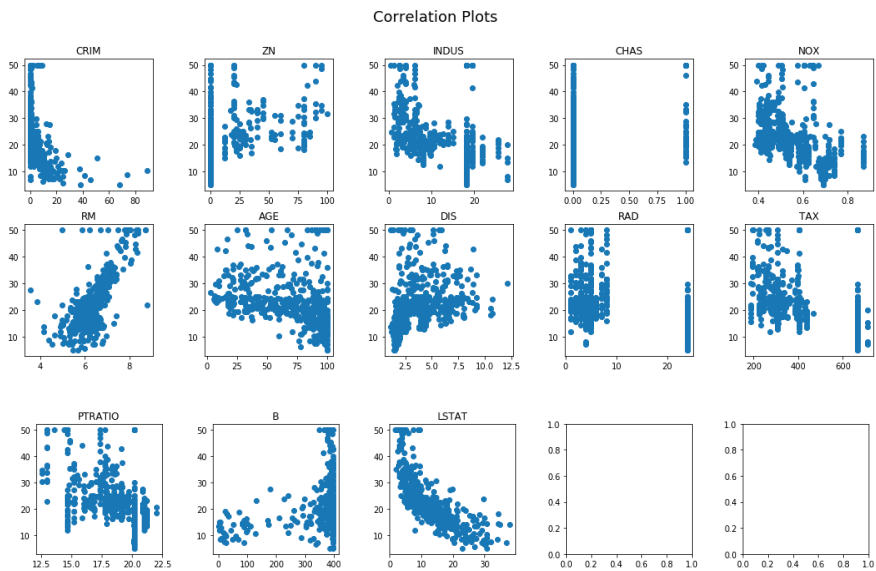
Correlation measures the statistical relationship between two variables. It can be negative or positive between –1 and 1.

- If the value of one variable increases with an increase in the value of the other variable, then the two variables are said to be positively correlated. (+1 means a strong positive correlation.)
- If the value of one variable decreases with an increase in the value of the other variable, then the two variables are said to be negatively correlated. (-1 means strong negative correlation.)
- If an increase or decrease in one variable does not affect the value of the other variable, then the two variables do not have any linear correlation (correlation = 0).

PYTHON CODE:

```
# Import pyplot from matplotlib to plot graphs

import matplotlib.pyplot as plt

# X represents the data and Y represents the respective labels

X=dataset.data
Y=dataset.target
row,col=dataset.data.shape

# subplots: 3 rows and 5 column (grid)
fig,ax = plt.subplots(3,5)

# Set height and width of figure in inches
fig.set_size_inches(18.25, 10.25)
```

```
# To adjust spacing and gap between subplots
plt.subplots_
adjust(top=0.9,hspace=0.25,wspace=0.4,bottom=0.15)

# Set title of complete figure

ig.suptitle('Correlation Plots',fontsize=18)

c_idx=0
for i in range(3):
    for j in range(5):
        ax[i,j].scatter(X[:,c_idx],Y)
        ax[i,j].set_title(dataset.feature_names[c_idx])
        c_idx+=1

        # We only have 13 features. So the loop should exit
after plotting them

        if(c_idx>=13):
          break
plt.show()
```



As you can see in the scatter plots, only RM and LSTAT seem to have a positive and negative correlation with home prices, respectively. We will only be using RM and LSTAT as our parameters if we are to build a linear model. Therefore, we need a nonlinear model to use all of our parameters and get good results. We will use a fully connected neural network model here, which can automatically learn weights to be assigned to each parameter in our model and can model nonlinearity in data.

8.1.2. Data Normalization

As our input features have a different range of values, it affects the training of a model. We first need to normalize all of our features. We normalize each feature by dividing its values with the maximum possible value for that feature. Then, we split the dataset into train, test, and validation.

PYTHON CODE:

```
# To split dataset into train and test split

from sklearn.model_selection import train_test_split

# Import numpy library
import numpy as np

x=dataset.data
y=dataset.target

# Iterating through parameters and finding maximum, minimum
values

for i,(minval,maxval) in enumerate(zip(np.amin(x,axis=0),np.
amax(x,axis=0))):
```

```
  # Dividing each parameter values by maximum possible value
for that parameter

  x[:,i]=x[:,i]/maxval

  # Splitting dataset into train and test set
  x_train,x_test,y_train,y_test=train_test_split(x,y,test_
size=0.1,random_state=7)

# Splitting training set into train and validation set

x_train,x_val,y_train,y_val=train_test_split(x,y,test_
size=0.1,random_state=43)

print()
print('x_train shape:',x_train.shape)
print('y_train shape:',y_train.shape)
print('x_val shape:',x_val.shape)
print('y_val shape:',y_val.shape)
print('x_test shape:',x_test.shape)
print('y_test shape:',y_test.shape)
```

```
x_train shape: (455, 13)
y_train shape: (455,)
x_val shape: (51, 13)
y_val shape: (51,)
x_test shape: (51, 13)
y_test shape: (51,)
```

8.2. Model Building

Our data has 13 input features. So, our first dense layer (input layer) will have 13 units. The input layer will be followed by three intermediate dense layers and one output layer containing one single neuron. This neuron will have our output regression value. We will compile our model with MSE loss and will use the same to measure the goodness of fit.

PYTHON CODE:

```
# Import Dense submodule from layers.core module

from keras.layers.core import Dense

# Import sequential submodule from models module

from keras.models import Sequential

# To flatten out some input, we have Flatten module/layer

from keras.layers.core import Flatten

model=Sequential()

# Since our input will be our 13 features, so we need 13
neurons in input layer

model.add(Dense(13,use_bias=True,activation='relu'))

# A dense layer wth 1024 output units
model.add(Dense(1024,use_bias=True,activation='relu'))

# A dense layer with 1024 output units
model.add(Dense(1024,use_bias=True,activation='relu'))

# A dense layer with 10 output units
model.add(Dense(10,use_bias=True,activation='relu'))
model.add(Dense(1,use_bias=True))

model.compile(loss='mean_squared_
error',optimizer='rmsprop',accuracy=['mse','mae'])
```

8.3. Model Training

We will train our model for 100 epochs with batch_size = 16 and will store the model weights in the checkpoints folder.

PYTHON CODE:

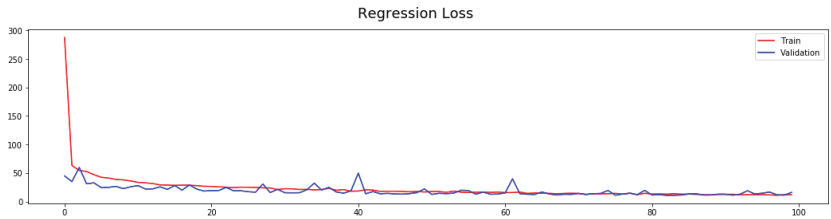
```
# To create checkpoints after finishing each epoch

from keras.callbacks import ModelCheckpoint
import os

checkpts='checkpts/'

# Create checkpoints directory if it does not exist

if(not os.path.exists(checkpts)):
  os.mkdir(checkpts)


#Following line creates a call back to store model weights
as checkpoints after every iteration. It will only store the
model weights which will have lowest val_loss value.

mcp = ModelCheckpoint(os.path.join(checkpts,'bestweights'),
                      monitor="val_loss",save_best_only=True,
save_weights_only=False)

# This fit function trains the model and returns the timely
changes in accuracy over time

hist=model.fit(x=x_train,y=y_train, epochs=100,
               validation_data=(x_val, y_
val),shuffle=True,batch_size=16,callbacks=[mcp])
```

8.4. Training and Validation Loss Plot

Plotting the history of model training:

PYTHON CODE:

```
# Import pyplot submodule from matplotlib
import matplotlib.pyplot as plt

def summarize_stats(history):
  # loss
  # A figure with a specific size
  plt.figure(figsize=(18,4))

  # Subplot containing one image
  plt.subplot(1,1,1)

  # This figure will show regression loss

  plt.suptitle('Regression Loss',fontsize=18)
  plt.plot(history.history['loss'], color='red',
label='Train')
  plt.plot(history.history['val_loss'], color='blue',
label='Validation')
  plt.legend(loc='upper right')

  # accuracy

  plt.legend(loc='upper right')
  plt.show()


summarize_stats(hist)
```



This graph tells us that our training and validation sets performed equally well once trained for 100 epochs. Now, we will test our model on test data.

8.5. Model Evaluation on Test Data

First, we will load stored weights. Then, we will evaluate our model's performance on test data.

PYTHON CODE:

```
# Load checkpoints from 'bestweights' folder

model.load_weights(os.path.join(checkpts,'bestweights'))

# Evaluate the same model with test set

out=model.evaluate(x_test,y_test)
print('MSE for test set:',out)
```

```
51/51 [==============================] - 0s 206us/step
MSE for test set: 8.784160595314175
```

8.6. Model Predictions

Here is the comparison of actual prices and model predicted prices:

PYTHON CODE:

```
predictions=model.predict(x_test)

top_n_results=5

for pred,y in zip(predictions[:top_n_results],y_test[:top_n_
results]):
  print('Actual Price: ',y,', Predicted Price: ',pred[0])
```

```
Actual Price:  21.7 , Predicted Price:  22.864927
Actual Price:  18.5 , Predicted Price:  21.527649
Actual Price:  22.2 , Predicted Price:  21.320627
Actual Price:  20.4 , Predicted Price:  20.918919
Actual Price:  8.8 , Predicted Price:  8.065931
```

9

Project 3: Binary Classification

In this chapter, we will cover building a convolutional neural network to classify the images of cats and dogs. We have a dataset, which you can download directly on your desktop or run following commands to download it on Google colab. Following are the key topics covered in this chapter:

1. Dataset Preparation
2. Building a Convolutional Model from Scratch
3. Training Model
4. Evaluating Model
5. Transfer Learning and Fine Tuning

PYTHON CODE:

```
# We need to import multiple libraries as these are being used
in this chapter.

# Import os for file handling mostly

import os

 # Shutil, same as for file handling

import shutil
```

```
# Glob is being used for file handling too

import glob

# Import OpenCV
import cv2

# Import numpy library
import numpy as np

# Import Path from pathlib

from pathlib import Path
import matplotlib.pyplot as plt

# Train test split from model_selection module

from sklearn.model_selection import train_test_split

# get confusion matrix and classification report

from sklearn.metrics import confusion_matrix,classification_
report

import keras

# To convert to one-hot encoding vector

from keras.utils import to_categorical

# Import dropout submodule from keras.layers main module

from keras.layers import Dropout

# To sequentially adding layers

from keras.models import Sequential
from keras.layers.core import Activation,Flatten,Dense
```

```
# Import for data preprocessing. More specifally, it is used
in data augmentation

from keras.preprocessing.image import ImageDataGenerator
from keras.layers.convolutional import MaxPooling2D, Conv2D


# A function to create a directory if it doesn't exist.

def create_directory(path,remove=True):
  if(os.path.exists(path)):
    if(remove==False):
      return
    shutil.rmtree(path)
  os.mkdir(path)
```

9.1. Dataset Preparation

There are two ways of using data to train a model:

1. We can get data from the internet or anywhere and place it in files ordered in a way that is understood by Keras and is available to be read.
2. Another way is to use Keras datasets library to load data that have been pre-included into the library.

We will look at each of the two cases in this chapter. We will focus on the first way of getting data and feeding it into the model.

For our binary classification problem, we will use the Dogs and Cats dataset for Kaggle competition available at url:

`https://download.microsoft.com/download/3/E/1/3E1C3F21-ECDB-4869-8368-6DEBA77B919F/kagglecatsanddogs_3367a.zip`

9.1.1. Dataset Loading

Run the following cell to download the dataset and unzip it.

```
!wget https://download.microsoft.com/download/3/E/1/3E1C3F21-ECDB-
4869-8368-6DEBA77B919F/kagglecatsanddogs_3367a.zip
```

```
!unzip -qq kagglecatsanddogs_3367a.zip
```

9.1.2. Dataset Curation

We need to check for any corrupt images in our dataset, which might hinder our training process. This is called dataset curation. The following action cell curates or cleans the dataset by removing corrupt images.

PYTHON CODE:

```
# Opens an image to makes sure that image is not broken

def check_broken(imgfile):
  try:
    im=cv2.imread(str(imgfile))
    if(im is None):
      return True
    return False
  except:
    return True
```

```
# All the broken images will be moved to another place

folders=os.listdir('PetImages/')
files=[]
for folder in folders:
  subfolderpath=os.path.join('PetImages/',folder)

  for filepath in list(Path(subfolderpath).glob('*')):
    if(check_broken(filepath)):
      files.append(filepath)
      os.remove(filepath)
print(len(files),'corrupt files have been removed.')
```

We have two classes in our dataset:

- Cat (we will represent cat class with label 0)
- Dog (we will represent dog class with label 1)

```
[ ]  !echo "Number of Images for Cat: `ls PetImages/Cat | wc -l`"
     !echo "Number of Images for Dog: `ls PetImages/Dog | wc -l`"
```

```
Number of Images for Cat: 12476
Number of Images for Dog: 12470
```

9.1.3. Subset of Dataset Selection and Dataset Split

After dataset curation, we have 12476 instances of Cat class and 12470 instances of Dog class. We can either use the entire dataset or just a subset of the dataset. For simplicity, we will use 5000 images only. But to see the complete results of the model, you should use all the images: 12476 + 12470. You can set the variable ***images_to_use = 24946***

After selecting images, we will also split our data into train, test, and validation sets.

PYTHON CODE:

```
# Only use 5000 images from data (to shorten the training
time)

images_to_use=5000

#following two lines will give us complete paths of cats and
dogs images

# Get paths to all cat images
catimages=np.array(glob.glob('PetImages/Cat/*'))

# Get paths to all dog images
dogimages=np.array(glob.glob('PetImages/Dog/*'))

# Randomly selected indices of images

random_selection=np.random.choice(len(catimages),images_to_
use//2)

# Cat images of randomly selected indices
catimages=catimages[random_selection].tolist()

# Randomly selected indices of dog images

random_selection=np.random.choice(len(dogimages),images_to_
use//2)

# Dog images of randomly selected indices
dogimages=dogimages[random_selection].tolist()



#cat images are numerically labeled as 0 and dog images are
labeled as 1

# cat images labeled as
catlabels=[0]*len(catimages)
```

```
# Dog images labeled as 1
doglabels=[1]*len(dogimages)

#We will combine cat and dog images to build our complete
dataset.

# Combine all images
all_images=catimages+dogimages

# Combine all labels
all_labels=catlabels+doglabels

print('all images length:',len(all_images))
print('all labels length:',len(all_labels))

# 80 - 20 train-test split for our data

x_train, x_test, y_train, y_test = train_test_split(all_
images, all_labels, test_size=0.2, random_state=43)

# we will further divide this train data into train data and
validation data with 90 - 10 split:

x_train,x_val,y_train,y_val=train_test_split(x_train,y_
train,test_size=0.1,random_state=7)

print('train set size: ',len(x_train))
print('test set size: ',len(x_test))
print('validation set size: ',len(x_val))
```

```
# 80 - 20 train-test split for our data
x_train, x_test, y_train, y_test = train_test_split(all_images, all_labels, test_size=0.2, random_state=43)

# we will further divide this train data into train data and validation data with 90 - 10 split:
x_train,x_val,y_train,y_val=train_test_split(x_train,y_train,test_size=0.1,random_state=7)

print('train set size: ',len(x_train))
print('test set size: ',len(x_test))
print('validation set size: ',len(x_val))
```

```
all images length: 5000
all labels length: 5000
train set size:  3600
test set size:  1000
validation set size:  400
```

9.1.4. Dataset Organization

We will organize train, test and validation data into a form that is recognized by Keras. For classification problem, we need create train, test and validation directories which would contain the respective data. Then inside each directory, we will have folders with the names of classes, and each of that folder will have images inside it.

PYTHON CODE:

```
# Now, we need to create separate directories for train,
#test and validation data and place images in respective
folders Just to make the process convenient, the arrays are
enclosed

#together in an array
X=[x_train,x_test,x_val]

# Labels are enclosed together in array
Y=[y_train,y_test,y_val]

C1=['train','test','val']
C2=['Cat','Dog']

#create new directory to store train, test and validation
images for each category and create separate folder for each
class (Cat and Dog)

create_directory('data')

for c1 in C1:
  create_directory(os.path.join('data',c1))
  for c2 in C2:
    create_directory(os.path.join('data',c1,c2))
```

```
#Copy files to new directories

for i,(x_data,y_data) in enumerate(zip(X,Y)):
  print('Copying',C1[i],'files')
  for img,label in zip(x_data,y_data):
    shutil.copy(img,os.path.join('data/',C1[i],C2[label]))
```

```
Copying train files
Copying test files
Copying val files
```

```
[ ]
    !echo "Number of Cat Images for Training set: `ls data/train/Cat | wc -l`"
    !echo "Number of Dog Images for Training set: `ls data/train/Dog | wc -l`"

    !echo "Number of Cat Images for Validation set: `ls data/val/Cat | wc -l`"
    !echo "Number of Dog Images for Validation set: `ls data/val/Dog | wc -l`"

    !echo "Number of Cat Images for Test set: `ls data/test/Cat | wc -l`"
    !echo "Number of Dog Images for Test set: `ls data/test/Dog | wc -l`"
```

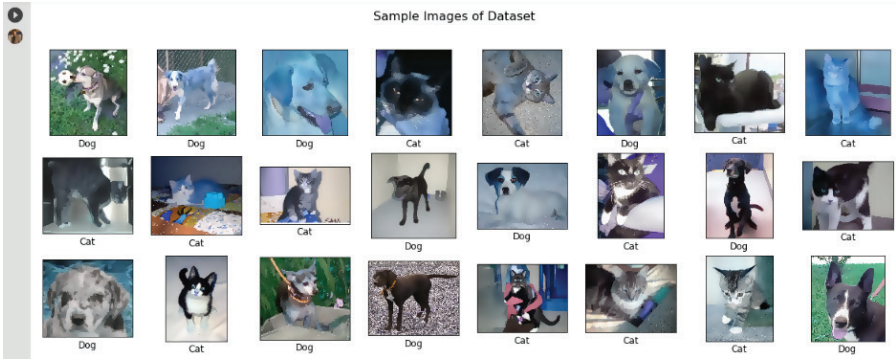
```
Number of Cat Images for Training set: 179
Number of Dog Images for Training set: 180
Number of Cat Images for Validation set: 24
Number of Dog Images for Validation set: 16
Number of Cat Images for Test set: 47
Number of Dog Images for Test set: 53
```

9.1.5. Dataset Visualization

Let's visualize a few samples from our training data.

PYTHON CODE:

```
# Initializing a figure of 20 x 7 in size

plt.figure(figsize=(20,7))
plt.suptitle('Sample Images of Dataset',fontsize=16)

# Have multiple subplot images and randomly insert cats and
dogs images

for i in range(24):
    plt.subplot(3,8,i+1)
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])

    # Show image in the subplot
    plt.imshow(cv2.imread(x_train[i]), cmap='gray')

    # Set x label of each subplot image
    plt.xlabel(C2[y_train[i]],fontsize=12)

plt.show()
```



9.2. Building Convolutional Neural Network

We can use some predefined CNN architectures, e.g., AlexNet, LeNet, VGG, etc. But for our understanding, we will build a simple convolutional network that can learn the patterns to differentiate between cats and dogs.

PYTHON CODE:

```
# Function to build a convolutional neural network

def build_CNN_model(inputshape):

  # Sequential type to build model by adding layers
  model=Sequential()

  # Convolutional layer with 64 kernels of size (3 x 3)
  model.add(Conv2D(64,(3,3),input_shape=inputshape))

  # Relu activation layer
  model.add(Activation('relu'))

  # Convolutional layer with 64 kernels of size (3 x 3)
  model.add(Conv2D(64,(3,3)))

  # Relu activation layer
  model.add(Activation('relu'))

  # Maxpooling layer with kernel = 2 x 2 shape
  model.add(MaxPooling2D(pool_size=2,strides=2))

  # Convolutional layer with 64 kernels of size (3 x 3)
  model.add(Conv2D(64,(3,3)))

  # Relu activation layer
  model.add(Activation('relu'))

  # Convolutional layer with 64 kernels of size (3 x 3)
  model.add(Conv2D(64,(3,3)))
```

```
# Relu activation layer
model.add(Activation('relu'))

# Maxpooling layer with kernel = 2 x 2 shape
model.add(MaxPooling2D(pool_size=2,strides=2))

# Convolutional layer with 64 kernels of size (3 x 3)
model.add(Conv2D(32,(3,3)))

# Relu activation layer
model.add(Activation('relu'))

# Convolutional layer with 64 kernels of size (3 x 3)
model.add(Conv2D(32,(3,3)))

# Relu activation layer
model.add(Activation('relu'))

# Maxpooling layer with kernel = 2 x 2 shape
model.add(MaxPooling2D(pool_size=2,strides=2))

#this will convert all the pixels of image to flat tensor
model.add(Flatten())

#add some fully connected layers

# Fully connected layer with 1024 neurons
model.add(Dense(1024))

# Relu activation layer
model.add(Activation('relu'))

# Dropout layer with dropout rate of 0.2
model.add(Dropout(0.2))

# Fully connected dense layer with 512 neurons
model.add(Dense(512))

# Relu activation layer
model.add(Activation('relu'))
```

```
  # Dropout layer with dropout rate of 0.2
  model.add(Dropout(0.2))

  #a fully connected layer with 2 neurons that represent the
probability to belong to Cat or Dog
  model.add(Dense(2))

  #a softmax to get probabilities of each class
  model.add(Activation('softmax'))

  return model
```

9.2.1. Model Summary

PYTHON CODE:

```
# Summary of the model with input shape = (224 x 224 x 3)

build_CNN_model(inputshape=(224,224,3)).summary()
```

9.3. Training

We will prepare variables and will setup directories for training our model. This includes creating directories for storing weights and logs. Moreover, we can set the callback functions that will be called during training after a fixed number of epochs.

PYTHON CODE:

```
# Use batchsize of 32 (Train the model with 32 input images as
one batch)

batchsize=32

# Train the model for 10 epochs
epochs=10

# Directory to store non augmented weights

nonaugweightsdir='nonaugcheckpoints/'
```

```
# Directory to store augmented weights

augweightsdir='augcheckpoints/'

# Directory to store logs

logdir='logs/'
categories=['Without Augmentation','With Augmentation']

#save model weights after every 2 epochs.

saveafter=2

# Create directory for storing model weights with non-
augmented data

create_directory(nonaugweightsdir,remove=False)

# Create directory for storing model weights with augmented
data

create_directory(augweightsdir,remove=False)

# Create directory for storing log

create_directory(logdir,remove=False)
```

9.4. Data Generator

We can manually write the code to load images into memory and feed to our convolutional model, but we will use ImageDataGenerator class provided by Keras. Using this class, we can iterate through our images in batches and feed them to our model. Besides image loading, this class provides the functionality to preprocess images (e.g., pixel scaling and augmentation). This also allows us only to load images

while feeding to the model. This way, we avoid storing a large number of images in memory as compared to some other methods available, e.g., using data from the Keras library.

Our convolutional network model takes fixed-size input images while our dataset contains images of different shapes. We need to set a fixed input size and resize all the images to that size. Let's use 224 x 224 x 3 dimensions for input image.

Finally, we will build two data generator types:

1. Without any augmentation.
2. With data augmentation: shift, rotation, shear, zoom, horizontal, and vertical flip.

We will also be able to see how augmentation can help learn the patterns in image variations.

PYTHON CODE:

```
# Initialize instance of data generator class

# Without augmentation - Normalize the image pixel values
nonauggenerator=ImageDataGenerator(rescale=1.0/255.0)

# With augmentation - includes rotation, shift, zoom , shear
and flip

auggenerator=ImageDataGenerator(rotation_range=30,
rescale=1./255, width_shift_range=0.2, height_shift_range=0.2,
zoom_range=0.2, shear_range=0.2, horizontal_flip=True, fill_
mode='nearest')

generators=[nonauggenerator,auggenerator]
```

```
# Three iterators: for train, test and validation data and
each iterator type has two entities (one for augmented data
and other for nonaugmented)

training_iterators=[]
testing_iterators=[]
validation_iterators=[]

# Use train, test and val data generators
for generator in generators:
  training_iterators.append(generator.flow_from_
directory('data/train/',target_size=(224,224),class_
mode='categorical',batch_size=batchsize))
  testing_iterators.append(generator.flow_from_directory('data/
test/',target_size=(224,224),class_mode='categorical',batch_
size=batchsize))
  validation_iterators.append(generator.flow_from_
directory('data/val/',target_size=(224,224),class_
mode='categorical',batch_size=batchsize))
```

```
Found 359 images belonging to 2 classes.
Found 100 images belonging to 2 classes.
Found 40 images belonging to 2 classes.
Found 359 images belonging to 2 classes.
Found 100 images belonging to 2 classes.
Found 40 images belonging to 2 classes.
```

Checkpoint: The model weights will be stored after n epoch (n is defined by saveafter variable above). This will allow us to retrieve the model weights later, for further training or model evaluation.

PYTHON CODE:

```
# Checkpoints callbacks for model with augmented data and with
non augmented data

nonaugcheckpoints=keras.callbacks.ModelCheckpoint(os.path.
join(nonaugweightsdir,"weights.{epoch:02d}-{val_loss:.2f}.
hdf5"), monitor='val_loss', verbose=0, save_best_only=False,
save_weights_only=False, mode='auto', period=saveafter)

augcheckpoints=keras.callbacks.ModelCheckpoint(os.path.
join(augweightsdir,"weights.{epoch:02d}-{val_loss:.2f}.hdf5"),
monitor='val_loss', verbose=0, save_best_only=False, save_
weights_only=False, mode='auto', period=saveafter)

checkpoints=[nonaugcheckpoints,augcheckpoints]
```

9.5. Training Convolutional Neural Network

We will initialize the convolutional neural network that we have already built to classify dogs and cats images. We will train the model for a few number of epochs and will analyze its results.

PYTHON CODE:

```
#we need to build two models, one for each data (with
augmentation and without augmentation)

models=[build_CNN_model(inputshape=(224,224,3)),build_CNN_
model(inputshape=(224,224,3))]
model_outputs=[]

for it,model in enumerate(models):
  print('\nTraining',categories[it])

  # Compile model (configure loss function, metrics and
optimizer)

  model.compile(loss='binary_crossentropy',
metrics=['accuracy'],optimizer='rmsprop')
```

```
  # Store model progress as hist object
  model_outputs.append(model.fit_generator(training_
iterators[it], steps_per_epoch=len(training_iterators[it]),

verbose=1,epochs=epochs,validation_data=validation_
iterators[it],
                                          validation_
steps=len(validation_iterators[it]),shuffle=True,

callbacks=[checkpoints[it]]))
```

```
W0707 08:52:18.262475 139932930942848 deprecation_wrapper.
py:119] From /usr/local/lib/python3.6/dist-packages/keras/
optimizers.py:790: The name tf.train.Optimizer is deprecated.
Please use tf.compat.v1.train.Optimizer instead.

W0707 08:52:18.286149 139932930942848 deprecation_wrapper.
py:119] From /usr/local/lib/python3.6/dist-packages/keras/
backend/tensorflow_backend.py:3376: The name tf.log is
deprecated. Please use tf.math.log instead.

W0707 08:52:18.293701 139932930942848 deprecation.py:323] From
/usr/local/lib/python3.6/dist-packages/tensorflow/python/ops/
nn_impl.py:180: add_dispatch_support.<locals>.wrapper (from
tensorflow.python.ops.array_ops) is deprecated and will be
removed in a future version.

Instructions for updating:

Use tf.where in 2.0, which has the same broadcast rule as
np.where
```

```
Training Without Augmentation
Epoch 1/10
12/12 [==============================] - 176s 15s/step - loss: 7.2163 - acc: 0.5063 - val_loss: 9.6181 - val_acc: 0.4000
Epoch 2/10
12/12 [==============================] - 171s 14s/step - loss: 8.3382 - acc: 0.4798 - val_loss: 9.6181 - val_acc: 0.4000
Epoch 3/10
12/12 [==============================] - 171s 14s/step - loss: 8.3382 - acc: 0.4798 - val_loss: 9.6181 - val_acc: 0.4000
Epoch 4/10
12/12 [==============================] - 171s 14s/step - loss: 8.0629 - acc: 0.4970 - val_loss: 9.6181 - val_acc: 0.4000
Epoch 5/10
12/12 [==============================] - 172s 14s/step - loss: 8.0629 - acc: 0.4970 - val_loss: 9.6181 - val_acc: 0.4000
Epoch 6/10
12/12 [==============================] - 172s 14s/step - loss: 7.9253 - acc: 0.5056 - val_loss: 9.6181 - val_acc: 0.4000
Epoch 7/10
12/12 [==============================] - 172s 14s/step - loss: 7.7877 - acc: 0.5142 - val_loss: 9.6181 - val_acc: 0.4000
Epoch 8/10
12/12 [==============================] - 172s 14s/step - loss: 8.0629 - acc: 0.4970 - val_loss: 9.6181 - val_acc: 0.4000
Epoch 9/10
12/12 [==============================] - 172s 14s/step - loss: 8.0629 - acc: 0.4970 - val_loss: 9.6181 - val_acc: 0.4000
Epoch 10/10
12/12 [==============================] - 171s 14s/step - loss: 7.7877 - acc: 0.5142 - val_loss: 9.6181 - val_acc: 0.4000
```

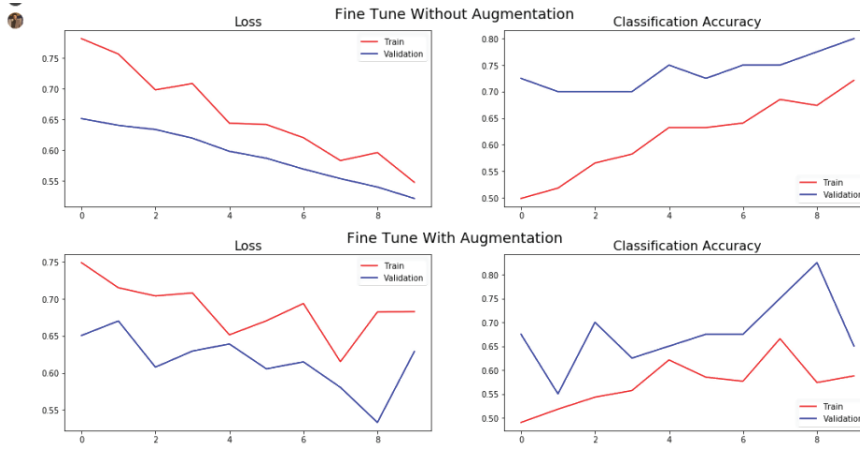
```
Training With Augmentation
Epoch 1/10
12/12 [==============================] - 183s 15s/step - loss: 7.7440 - acc: 0.4682 - val_loss: 9.6181 - val_acc: 0.4000
Epoch 2/10
12/12 [==============================] - 175s 15s/step - loss: 8.2006 - acc: 0.4884 - val_loss: 9.6181 - val_acc: 0.4000
Epoch 3/10
12/12 [==============================] - 175s 15s/step - loss: 7.7877 - acc: 0.5142 - val_loss: 9.6181 - val_acc: 0.4000
Epoch 4/10
12/12 [==============================] - 176s 15s/step - loss: 8.0629 - acc: 0.4970 - val_loss: 9.6181 - val_acc: 0.4000
Epoch 5/10
12/12 [==============================] - 176s 15s/step - loss: 7.6501 - acc: 0.5228 - val_loss: 9.6181 - val_acc: 0.4000
Epoch 6/10
12/12 [==============================] - 176s 15s/step - loss: 8.3382 - acc: 0.4798 - val_loss: 9.6181 - val_acc: 0.4000
Epoch 7/10
12/12 [==============================] - 177s 15s/step - loss: 8.2006 - acc: 0.4884 - val_loss: 9.6181 - val_acc: 0.4000
Epoch 8/10
12/12 [==============================] - 177s 15s/step - loss: 8.0629 - acc: 0.4970 - val_loss: 9.6181 - val_acc: 0.4000
Epoch 9/10
12/12 [==============================] - 177s 15s/step - loss: 8.0629 - acc: 0.4970 - val_loss: 9.6181 - val_acc: 0.4000
Epoch 10/10
12/12 [==============================] - 175s 15s/step - loss: 8.0629 - acc: 0.4970 - val_loss: 9.6181 - val_acc: 0.4000
```

9.5.1. Visualizing Training Loss and Accuracies

PYTHON CODE:

```
# Plot the progress/stats of the model

def summarize_stats(history,title):

  # loss
  # Initialize a figure
  plt.figure(figsize=(18,4))

  # Initialize subplots
  plt.subplot(1,2,1)

  # Title of complete image
  plt.suptitle(title,fontsize=18)

  # Title of subplot
  plt.title('Loss',fontsize=16)

  # Plot the training loss

  plt.plot(history.history['loss'], color='red',
label='Train')
```

```
  # Plot the validation loss

  plt.plot(history.history['val_loss'], color='blue',
label='Validation')

  # Put legends on top right corner
  plt.legend(loc='upper right')

  # accuracy
  plt.subplot(1,2,2)

  # Set title of second subplot
  plt.title('Classification Accuracy',fontsize=16)

  # Plot training accuracy

  plt.plot(history.history['acc'], color='red', label='Train')

  # Plot validation accuracy

  plt.plot(history.history['val_acc'], color='blue',
label='Validation')

  plt.legend(loc='lower right')
  plt.show()

for it,model in enumerate(models):

  # For all the types of models, plot hist object stats
  summarize_stats(model_outputs[it],title=str(categories[it]))
```



9.6. Testing Model Weights

The weights of the model were stored in **nonaugcheckpoints/** directory and **augcheckpoints/** directory, after every two epochs. We will list down those weights files.

PYTHON CODE:

```
print('\nModel for Non Augmented Data: ')
print(os.listdir('nonaugcheckpoints/'))
print('\nModel for Augmented Data: ')
print(os.listdir('augcheckpoints/'))
```

```
Model for Non Augmented Data:
['weights.10-6.41.hdf5', 'weights.04-0.70.hdf5', 'weights.08-6.41.hdf5', 'weights.06-0.78.hdf5', 'weights.02-0.70.hdf5']

Model for Augmented Data:
['weights.04-0.69.hdf5', 'weights.10-0.70.hdf5', 'weights.06-0.69.hdf5', 'weights.02-0.68.hdf5', 'weights.08-0.69.hdf5']
```

9.6.1. Weight Loading

For our CNN model, we will load weights from the file and evaluate on test data. We can load from any weight file stored in directory **checkpoints**/

PYTHON CODE:

```
# Load model weights trained with non augmented data

models[0].load_weights(glob.glob('nonaugcheckpoints/*')[-1])

# Load model weights trained with augmented data

models[1].load_weights(glob.glob('augcheckpoints/*')[-1])

for it,model in enumerate(models):
  print(categories[it])

  # Evaluate model on test data

  test_loss, accuracy = model.evaluate_generator(testing_
iterators[it],
  len(testing_iterators[it]), verbose=1)

  # Print evaluation loss and accuracy

  print('Evaluation loss: %.2f, Accuracy: %.2f' % (test_
loss,accuracy * 100.0))
```

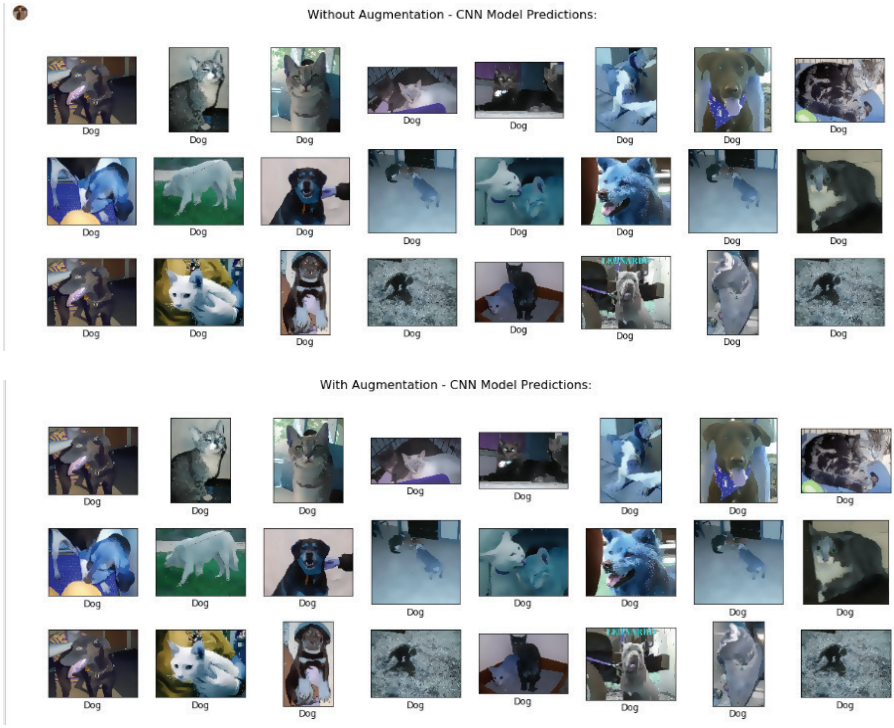
```
Without Augmentation
7/7 [==============================] - 12s 2s/step
Evaluation loss: 0.69, Accuracy: 53.00
With Augmentation
7/7 [==============================] - 15s 2s/step
Evaluation loss: 0.69, Accuracy: 53.00
```

9.6.2. Visualizing Predictions

PYTHON CODE:

```
# Do for each type of model

predictions=[]
for it,model in enumerate(models):
  print(categories[it])
```

```
  # Get predictions for each type of model
  predictions.append(model.predict_generator(testing_
iterators[it],len(testing_iterators[it])))

random_selection=np.random.choice(len(x_test),24)

# For all the predictions from model

for it,prediction in enumerate(predictions):

  # Initialize a figure
  plt.figure(figsize=(20,7))
  plt.suptitle(str(categories[it])+' - CNN Model
Predictions:',fontsize=16)

  # 24 subplots in total

  for i in range(24):

      # Select a subplot
      plt.subplot(3,8,i+1)

      # Set the grid off (no gridlines)
      plt.grid(False)

      # No numbers in x axis
      plt.xticks([])

      # No numbers in y axis
      plt.yticks([])

      plt.imshow(cv2.imread(x_test[random_selection[i]]),
cmap='gray')
      plt.xlabel(str(C2[np.argmax(prediction[random_
selection[i]])]),fontsize=12)

  plt.show()
```



9.6.3. Confusion Matrix:

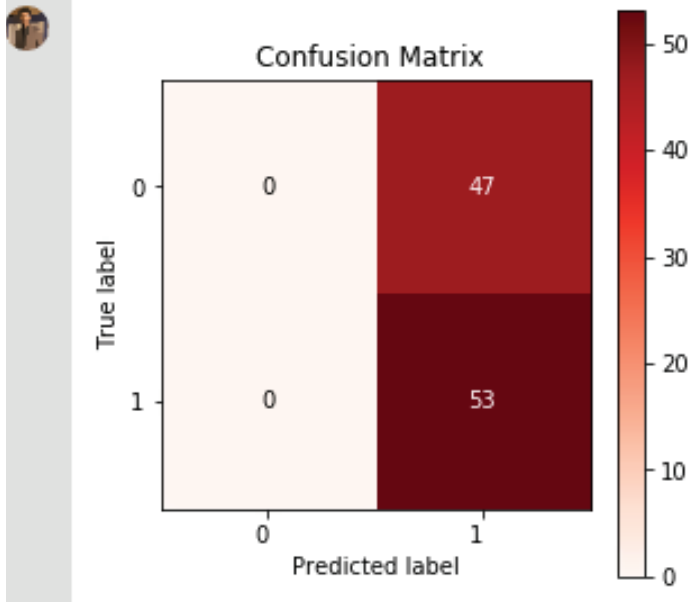
Confusion matrix evaluates the model to see which classes have been correctly classified most often and which classes are hard for the model to classify. The diagonal elements show for which the actual label of the image and model predicted label are the same, while off-diagonal elements show misclassified image classes.

PYTHON CODE:

```
def plot_confusion_matrix(y_true, y_pred, titletxt,
cmap=plt.cm.Reds):

    # Get confusion matrix based on prediction values and
actual values

    cm = confusion_matrix(y_true, y_pred)

    # As we only have two classes: cats and dogs

    classes = range(0,2)

    # Initialize a figure

    fig, ax = plt.subplots(figsize=(4,4))

    # Show confusion matrix with passed cmap

    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)

    # Choosing color bar

    ax.figure.colorbar(im, ax=ax)

    # We want to show all ticks...

    ax.set(xticks=np.arange(cm.shape[1]),
           yticks=np.arange(cm.shape[0]),
           xticklabels=classes, yticklabels=classes,
           title='Confusion Matrix ',
           ylabel='True label',
           xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.

    plt.setp(ax.get_xticklabels(), ha="right",
             rotation_mode="anchor")
```

```
    # Loop over data dimensions and create text annotations.

    fmt = 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else
"black")
    fig.tight_layout()
    return ax


# Convert labels to one hot encoding

# Get predicted label

one_hot_y_test=to_categorical(y_test,num_classes=2)

for it,prediction in enumerate(predictions):
  predicted=np.argmax(prediction,axis=1)

  # From one hot encoding to indexes
  actual=np.argmax(np.array(one_hot_y_test),axis=1)

  # Plot confusion matrix

  plot_confusion_matrix(actual,predicted,categories[it])
  print('\n',categories[it],'Accuracy Metrics for each
class:')

  # Print classification report

  print(classification_report(actual,predicted))
```

```
(100, 2)

 Without Augmentation Accuracy Metrics for each class:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        47
           1       0.53      1.00      0.69        53

    accuracy                           0.53       100
   macro avg       0.27      0.50      0.35       100
weighted avg       0.28      0.53      0.37       100


 With Augmentation Accuracy Metrics for each class:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        47
           1       0.53      1.00      0.69        53

    accuracy                           0.53       100
   macro avg       0.27      0.50      0.35       100
weighted avg       0.28      0.53      0.37       100
```



10

Project 4: Fine Tuning Pretrained Model—VGG16

We built a simple model consisting of a few convolutional, max pooling, and fully connected layers. We will now use VGG16 pretrained on ImageNet. Since this model has been trained on a huge dataset, we can use those parameters to initialize with and make the model learn for our specific problem. As we know, the four steps to fine-tune a model are:

- Load a pretrained model and set include_top=False to remove the last fully connected layers.
- Freeze initial layers because we do not want to train them.
- Build our own model on top of a pretrained one.
- Train the model.

10.1. Loading Pretrained Model

Since we do not want top dense layers, we will not use dense layers from VGG16. Instead, we will build our own model for our specific problem. Since our input image has the shape:

(224,224,3), we will pass this shape while initialiing the VGG16 model.

PYTHON CODE:

```
# Load VGG16 model

from keras.applications import VGG16

# Use imagenet weights of pretrained VGG16

vgg16=VGG16(weights='imagenet',input_
shape=(224,224,3),include_top=False)
```

10.2. Freezing the Initial Layers

The VGG16 model has five blocks, and each block has Convolutional and Max-pooling layers. We will freeze the first four blocks, and we will train the last block to learn the features relevant to solve our classification problem.

PYTHON CODE:

```
# Freeze all the layers of VGG except the last 4

for layer in vgg16.layers[:-4]:
  layer.trainable=False
```

10.3. Build a Model on Top of Base Model

This VGG model will act as our base model. Now, we will add our classifier on top of this base model. We will add the same last layers from our previous model.

PYTHON CODE:

```
# Import Sequential submodule of keras models module

from keras.models import Sequential

# Import Dense, Activation and flatten submodules from keras
layers.core module

from keras.layers.core import Dense, Activation, Flatten

# Import convolutional layer

from keras.layers.convolutional import Conv2D

# Import maxpooling layer of keras' layers module

from keras.layers import MaxPooling2D

# Import dropout layer from keras layers module

from keras.layers import Dropout

# Function to build a complete model for our task (based on a
base model)

def build_model_finetune(basemodel):
  model=Sequential()

  # Add base model
  model.add(basemodel)

  #this will convert all the pixels of image to flat tensor
  model.add(Flatten())

  # Fully connected layer with 1024 neurons
  model.add(Dense(1024))

  # Relu activation layer
  model.add(Activation('relu'))
```

```
  # Dropout layer with 0.2 dropout rate
  model.add(Dropout(0.2))

  # Fully connected layer with 512
  model.add(Dense(512))

  # Relu activation layer
  model.add(Activation('relu'))

  # Dropout layer with 0.2 dropout rate
  model.add(Dropout(0.2))

  #a fully connected layer with 2 neurons that represent the
probability to belong to Cat or Dog

  model.add(Dense(2))

  #a softmax to get probabilities of each class

  model.add(Activation('softmax'))

  return model

build_model_finetune(vgg16).summary()
```

10.4. Training the Model

As we trained our earlier model on the two data iterators, i.e., with augmentation and without augmentation, we will do the same for this model, too.

PYTHON CODE:

```
# Use 32 images to backprop and update model weights at once

batchsize=32

# Train model for 10 epochs

epochs=10

# Model trained with Non augmentated data checkpoints

nonaugweightsdir_ft='vgg_nonaug_checkpoints/'

# Model trained with augmented data checkpoints

augweightsdir_ft='vgg_aug_checkpoints/'
logdir='logs/'
categories=['Fine Tune Without Augmentation','Fine Tune With
Augmentation']

#save model weights after n number of epochs.
# Safe model weights after every 1 epoch

saveafter=1

# Create directory to store model weights with non augmented
data

create_directory(nonaugweightsdir_ft,remove=False)

# Create directory to store model weights with augmented data

create_directory(augweightsdir_ft,remove=False)

# Create directory to store log
create_directory(logdir,remove=False)
```

Checkpoint: The model weights will be stored after n epochs (n is defined by the saveafter variable above). This will allow us to retrieve the model weights later, for further training or model evaluation.

PYTHON CODE:

```
# Checkpoints call back for model trained with non augmented
data

nonaugcheckpoints=keras.callbacks.ModelCheckpoint(os.path.
join(nonaugweightsdir_ft,"weights.{epoch:02d}-{val_loss:.2f}.
hdf5"),monitor='val_loss', verbose=0, save_best_only=False,
save_weights_only=False, mode='auto', period=saveafter)

# Checkpoints call back for model trained with augmented data

augcheckpoints=keras.callbacks.ModelCheckpoint(os.path.
join(augweightsdir_ft,"weights.{epoch:02d}-{val_loss:.2f}.
hdf5"),monitor='val_loss', verbose=0, save_best_only=False,
save_weights_only=False, mode='auto', period=saveafter)

checkpoints=[nonaugcheckpoints,augcheckpoints]

# Import RMSprop optimizer from keras.optimizers module

from keras.optimizers import RMSprop

models=[build_model_finetune(vgg16),build_model_finetune(vgg16)]

model_outputs=[]

for it,model in enumerate(models):
  print('\nTraining',categories[it])

  # Compile the model (configure loss function, metrics and
optimizer)
  model.compile(loss='categorical_
crossentropy',metrics=['accuracy'],optimizer=RMSprop(lr=1e-6))
```

```
  # Store training history/stats for each model
  model_outputs.append(model.fit_generator(training_
iterators[it], steps_per_epoch=len(training_iterators[it]),
verbose=1,epochs=epochs,validation_data=validation_
iterators[it],validation_steps=len(validation_
iterators[it]),shuffle=True, callbacks=[checkpoints[it]]))
```

```
Training Fine Tune Without Augmentation
Epoch 1/10
12/12 [==============================] - 250s 21s/step - loss: 0.7807 - acc: 0.5030 - val_loss: 0.6514 - val_acc: 0.7250
Epoch 2/10
12/12 [==============================] - 247s 21s/step - loss: 0.7670 - acc: 0.5041 - val_loss: 0.6403 - val_acc: 0.7000
Epoch 3/10
12/12 [==============================] - 246s 21s/step - loss: 0.6843 - acc: 0.5916 - val_loss: 0.6336 - val_acc: 0.7000
Epoch 4/10
12/12 [==============================] - 246s 20s/step - loss: 0.7211 - acc: 0.5729 - val_loss: 0.6195 - val_acc: 0.7000
Epoch 5/10
12/12 [==============================] - 245s 20s/step - loss: 0.6437 - acc: 0.6287 - val_loss: 0.5981 - val_acc: 0.7500
Epoch 6/10
12/12 [==============================] - 246s 20s/step - loss: 0.6453 - acc: 0.6287 - val_loss: 0.5868 - val_acc: 0.7250
Epoch 7/10
12/12 [==============================] - 247s 21s/step - loss: 0.6158 - acc: 0.6537 - val_loss: 0.5691 - val_acc: 0.7500
Epoch 8/10
12/12 [==============================] - 247s 21s/step - loss: 0.5988 - acc: 0.6698 - val_loss: 0.5536 - val_acc: 0.7500
Epoch 9/10
12/12 [==============================] - 248s 21s/step - loss: 0.5946 - acc: 0.6679 - val_loss: 0.5398 - val_acc: 0.7750
Epoch 10/10
12/12 [==============================] - 250s 21s/step - loss: 0.5489 - acc: 0.7124 - val_loss: 0.5213 - val_acc: 0.8000
```

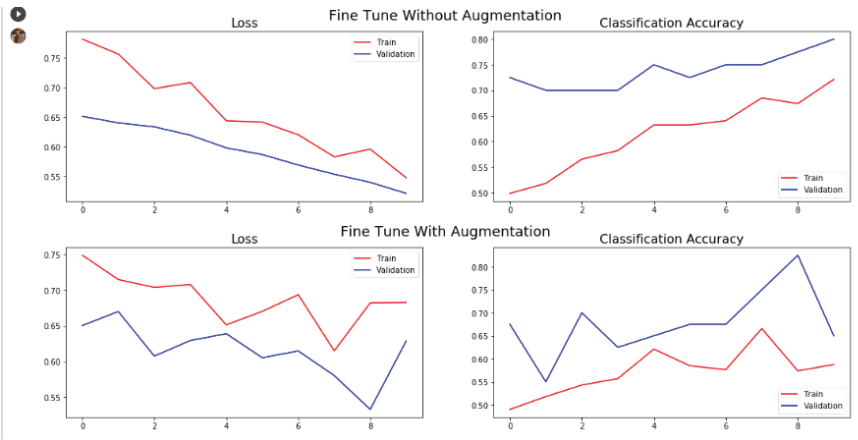
```
Training Fine Tune With Augmentation
Epoch 1/10
12/12 [==============================] - 259s 22s/step - loss: 0.7543 - acc: 0.4780 - val_loss: 0.6504 - val_acc: 0.6750
Epoch 2/10
12/12 [==============================] - 250s 21s/step - loss: 0.7215 - acc: 0.5127 - val_loss: 0.6700 - val_acc: 0.5500
Epoch 3/10
12/12 [==============================] - 254s 21s/step - loss: 0.7082 - acc: 0.5449 - val_loss: 0.6077 - val_acc: 0.7000
Epoch 4/10
12/12 [==============================] - 252s 21s/step - loss: 0.7046 - acc: 0.5665 - val_loss: 0.6294 - val_acc: 0.6250
Epoch 5/10
12/12 [==============================] - 251s 21s/step - loss: 0.6389 - acc: 0.6354 - val_loss: 0.6390 - val_acc: 0.6500
Epoch 6/10
12/12 [==============================] - 254s 21s/step - loss: 0.6783 - acc: 0.5756 - val_loss: 0.6054 - val_acc: 0.6750
Epoch 7/10
12/12 [==============================] - 252s 21s/step - loss: 0.6865 - acc: 0.5849 - val_loss: 0.6149 - val_acc: 0.6750
Epoch 8/10
12/12 [==============================] - 250s 21s/step - loss: 0.6148 - acc: 0.6687 - val_loss: 0.5805 - val_acc: 0.7500
Epoch 9/10
12/12 [==============================] - 251s 21s/step - loss: 0.6717 - acc: 0.5908 - val_loss: 0.5332 - val_acc: 0.8250
Epoch 10/10
12/12 [==============================] - 250s 21s/step - loss: 0.6867 - acc: 0.5868 - val_loss: 0.6290 - val_acc: 0.6500
```

10.4.1. Visualizing Training Loss and Accuracies

PYTHON CODE:

```
# Function to plot summary and statistics

def summarize_stats(history,title):

  # loss
  # Initialize a figure
  plt.figure(figsize=(18,4))
  plt.subplot(1,2,1)
  plt.suptitle(title,fontsize=18)

  # Set title of the first subplot
  plt.title('Loss',fontsize=16)

  # Plot training loss

  plt.plot(history.history['loss'], color='red',
label='Train')

  # Plot validation loss

  plt.plot(history.history['val_loss'], color='blue',
label='Validation')

  plt.legend(loc='upper right')

  # accuracy
  plt.subplot(1,2,2)

  # Set title of second subplot
  plt.title('Classification Accuracy',fontsize=16)

  # Plot training accuracy

  plt.plot(history.history['acc'], color='red', label='Train')
```

```
  # Plot validation accuracy

  plt.plot(history.history['val_acc'], color='blue',
label='Validation')

  plt.legend(loc='lower right')
  plt.show()

for it,model in enumerate(models):
  # Print summary for each model outputs history
  summarize_stats(model_outputs[it],title=str(categories[it]))
```



10.5. Testing Pretrained Model

10.5.1. Model Weights

The weights of the model were stored in the **vgg_nonaug_checkpoints/**directory and the **vgg_aug_checkpoints/** directory, after every two epochs. We will list down those weights files.

PYTHON CODE:

```
print('\nModel for Non Augmented Data: ')
print(os.listdir(nonaugweightsdir_ft))
print('\nModel for Augmented Data: ')
print(os.listdir(augweightsdir_ft))
```

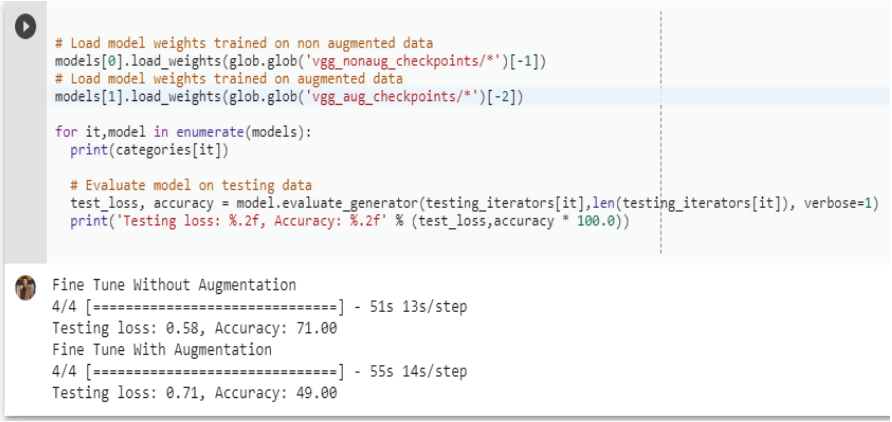
```
Model for Non Augmented Data:
['weights.06-0.59.hdf5', 'weights.02-0.64.
hdf5', 'weights.03-0.63.hdf5', 'weights.01-0.65.
hdf5', 'weights.07-0.57.hdf5', 'weights.05-0.60.
hdf5', 'weights.09-0.54.hdf5', 'weights.04-0.62.hdf5',
'weights.08-0.55.hdf5', 'weights.10-0.52.hdf5']

Model for Augmented Data:
['weights.03-0.61.hdf5', 'weights.04-0.63.
hdf5', 'weights.07-0.61.hdf5', 'weights.06-0.61.
hdf5', 'weights.08-0.58.hdf5', 'weights.09-0.53.
hdf5', 'weights.01-0.65.hdf5', 'weights.05-0.64.hdf5',
'weights.02-0.67.hdf5', 'weights.10-0.63.hdf5']
```

10.5.2. Weight Loading

For our convolutional model, we will load weights from file and evaluate on test data. We can load the weight file stored in the directory **vgg_nonaug_checkpoints/** or **vgg_aug_checkpoints/**

PYTHON CODE:

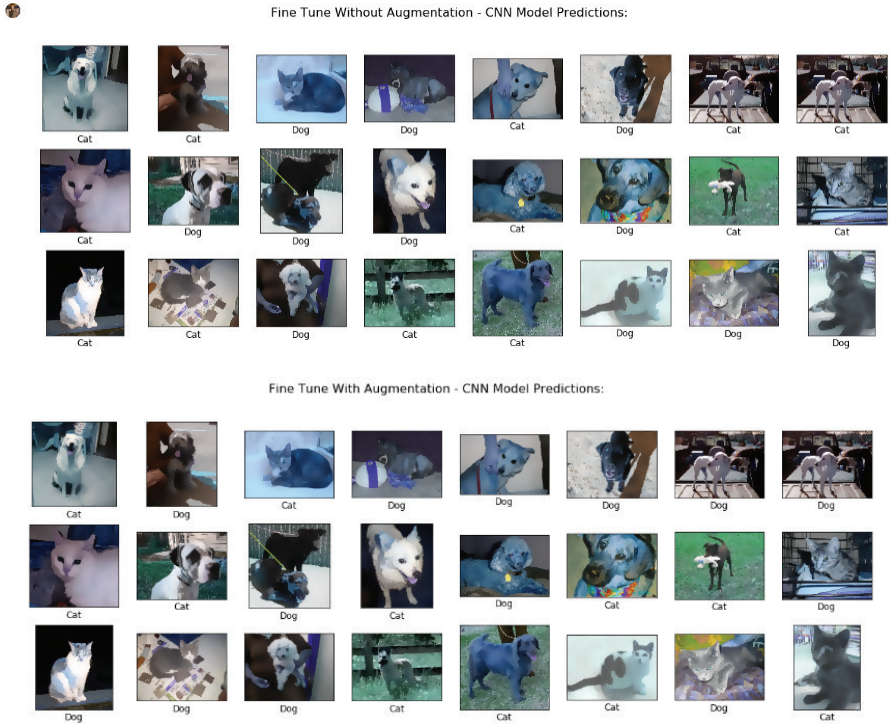
```
# Load model weights trained on non augmented data

models[0].load_weights(glob.glob('vgg_nonaug_checkpoints/*')
[-1])

# Load model weights trained on augmented data

models[1].load_weights(glob.glob('vgg_aug_checkpoints/*')[-2])

for it,model in enumerate(models):
  print(categories[it])

  # Evaluate model on testing data

  test_loss, accuracy = model.evaluate_generator(testing_
iterators[it],len(testing_iterators[it]), verbose=1)

  print('Testing loss: %.2f, Accuracy: %.2f' % (test_
loss,accuracy * 100.0))
```



10.5.3. Visualizing Predictions

PYTHON CODE:

```
predictions=[]
for it,model in enumerate(models):
  print(categories[it])

  # Store model predictions
  predictions.append(model.predict_generator(testing_
iterators[it],len(testing_iterators[it])))

random_selection=np.random.choice(len(x_test),24)

for it,prediction in enumerate(predictions):

  # Initialize a figure
  plt.figure(figsize=(20,7))

  plt.suptitle(str(categories[it])+' - CNN Model
Predictions:',fontsize=16)

  for i in range(24):

      # Initialize and select current subplot
      plt.subplot(3,8,i+1)

      # Set off the gridlines
      plt.grid(False)

      # Set off x axis values
      plt.xticks([])

      # Set off y axis values
      plt.yticks([])

      # Show the image in current subplot

      plt.imshow(cv2.imread(x_test[random_selection[i]]),
cmap='gray')
      plt.xlabel(str(C2[np.argmax(prediction[random_
selection[i]])]),fontsize=12)

  plt.show()
```



10.5.4. Confusion Matrix:

The confusion matrix will show how many instances from each class have been correctly or incorrectly classified from our fine-tuned model.

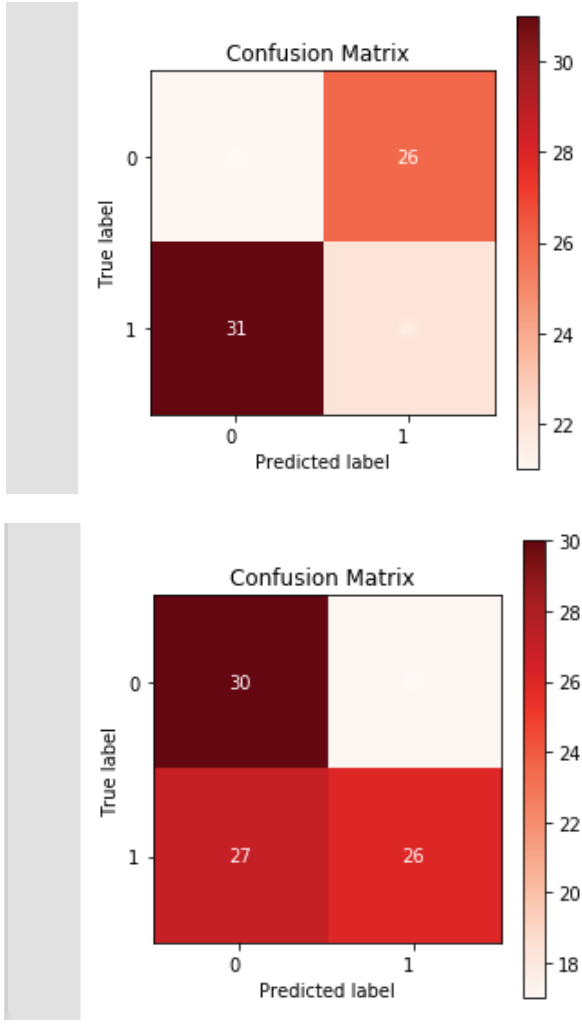
PYTHON CODE:

```
def plot_confusion_matrix(y_true, y_pred, titletxt, cmap=plt.
cm.Reds):

    # Get confusion matrix from actual y labels and predicted
y labels

    cm = confusion_matrix(y_true, y_pred)
```

```
    # Two classes: cats and dogs
    classes = range(0,2)

    fig, ax = plt.subplots(figsize=(4,4))

    # Show confusion matrix in axes/subplot

    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)

    # We want to show all ticks...

    ax.set(xticks=np.arange(cm.shape[1]),
           yticks=np.arange(cm.shape[0]),

        # ... and label them with the respective list entries

        xticklabels=classes, yticklabels=classes,
        title='Confusion Matrix ',
        ylabel='True label',
        # Set properties of axes/subplots
        xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.

    plt.setp(ax.get_xticklabels(), ha="right",
             rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.

    fmt = 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else
"black")
    fig.tight_layout()
    return ax
```

```
# Convert labels to one hot encoding to train the model

one_hot_y_test=to_categorical(y_test,num_classes=2)

print(one_hot_y_test.shape)

for it,prediction in enumerate(predictions):

  # Get predicted label
  predicted=np.argmax(prediction,axis=1)

  # Convert one hot encoded labels to indices
  actual=np.argmax(np.array(one_hot_y_test),axis=1)

  # Plot confusion matrix

  plot_confusion_matrix(actual,predicted,categories[it])
  print('\n',categories[it],'Accuracy Metrics for each
class:')

  # Print classification report
  print(classification_report(actual,predicted))
```

```
(100, 2)

 Fine Tune Without Augmentation Accuracy Metrics for each class:
              precision    recall  f1-score   support

           0       0.40      0.45      0.42        47
           1       0.46      0.42      0.44        53

    accuracy                           0.43       100
   macro avg       0.43      0.43      0.43       100
weighted avg       0.43      0.43      0.43       100


 Fine Tune With Augmentation Accuracy Metrics for each class:
              precision    recall  f1-score   support

           0       0.53      0.64      0.58        47
           1       0.60      0.49      0.54        53

    accuracy                           0.56       100
   macro avg       0.57      0.56      0.56       100
weighted avg       0.57      0.56      0.56       100
```



We can clearly see that a pretrained model is way better than a simple randomly initialized model. Most of the images have been correctly classified even when the model is trained on just 5000 images and for only 10 epochs. Training the models on all images will certainly improve the performance of the model by a large margin.

11

Project 5: Multiclass Classification

This chapter covers model building for multiclass classification on the MNIST digits dataset preloaded in the Keras library. More specifically, we will build neural networks to recognize handwritten digits (0 – 9). We will build two types of models:

1. Fully Connected Neural Network (Containing all dense layers)
2. Convolutional Neural Network (Containing Conv-Maxpool-Dense combination)

We will see and compare the performances of both models. Moreover, this chapter covers the VGG19 model pretrained on the imagenet dataset and fine-tuned on our digits dataset. So, we will also compare its results with our CNN.

This chapter walks you through dataset preparation, model development, model training, and finally, model evaluation using the Keras deep learning library.

PYTHON CODE:

```
import os
from tensorflow.keras.datasets import mnist
from sklearn.metrics import confusion_matrix, classification_
report
from sklearn.utils.multiclass import unique_labels
import collections
import numpy as np
import matplotlib.pyplot as plt
import keras
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers.core import Activation,Flatten,Dense
from keras.layers.convolutional import MaxPooling2D, Conv2D


#Using TensorFlow Backend

def create_directory(path):
  if(os.path.exists(path)):
    return
  os.mkdir(path)
```

11.1. Dataset Preparation

This time, we will use the in-built dataset library of Keras to load the MNIST dataset. We have ten classes in our dataset (0 – 9), one for each digit.

11.1.1. Dataset Loading

We have 60,000 images in the training set and 10,000 images in the test set. The number of instances in each class is almost the same. Since we already have a test and train split, we will split train data into train and validation split. Our inputs are grayscale images of 28 x 28 dimension, and outputs are labels

(digit in the image). We need to normalize those images and reshape them to match the required input shape for the Keras model. This isn't a very large dataset, so it can fit into memory.

PYTHON CODE:

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
print('Training set size: ',len(x_train))
print('Test set size: ',len(x_test))

distributions=[collections.Counter(y_train),collections.
Counter(y_test)]

for it,p in enumerate(['Training set distribution','Testing
set distribution']):
  print(p)
  for i in range(0,9):
    print('Digit: ',i,' | Frequency: ',distributions[it][i])
  print()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
```

```
Training set size:  60000
Test set size:  10000
Training set distribution
Digit:  0  | Frequency:  5923
Digit:  1  | Frequency:  6742
Digit:  2  | Frequency:  5958
Digit:  3  | Frequency:  6131
Digit:  4  | Frequency:  5842
Digit:  5  | Frequency:  5421
Digit:  6  | Frequency:  5918
Digit:  7  | Frequency:  6265
Digit:  8  | Frequency:  5851

Testing set distribution
Digit:  0  | Frequency:  980
Digit:  1  | Frequency:  1135
Digit:  2  | Frequency:  1032
Digit:  3  | Frequency:  1010
Digit:  4  | Frequency:  982
Digit:  5  | Frequency:  892
Digit:  6  | Frequency:  958
Digit:  7  | Frequency:  1028
Digit:  8  | Frequency:  974
```

11.1.2. Train — Validation Split

We will further divide the train set into a 90 percent train set and a 10 percent validation set using the sklearn library function.

PYTHON CODE:

```
from sklearn.model_selection import train_test_split

# we will divide this train data into train data and
validation data with 90 - 10 split:

x_train,x_val,y_train,y_val=train_test_split(x_train,y_
train,test_size=0.1,random_state=7)
```

11.1.3. Dataset Preparation

We will prepare the data to be used by the Keras model. This includes reshaping the data and normalizing the images. We can also use data augmentation, which includes random flip, shear, zoom, or rotation. We also need to convert the labels to the one-hot encoded form.

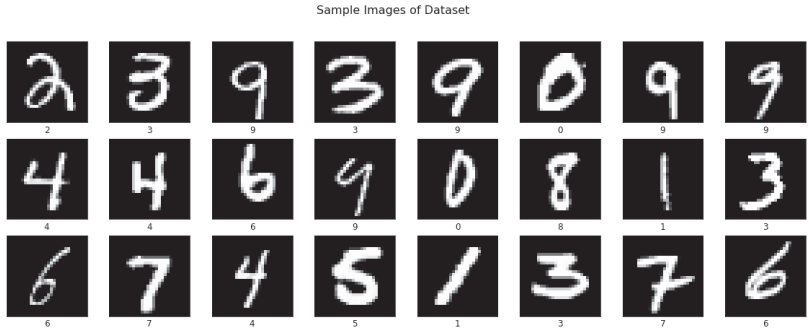
PYTHON CODE:

```
# Converting labels to one-hot encoded form
y_train=to_categorical(y_train,num_classes=10)
y_test=to_categorical(y_test,num_classes=10)
y_val=to_categorical(y_val,num_classes=10)

# We will reshape the array so that it can work with the Keras
API
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_val = x_val.reshape(x_val.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)


x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_val=x_val.astype('float32')

# Normalizing images
x_train /= 255
x_val/=255
x_test /= 255


print('train set: ',x_train.shape)
print('val set: ',x_val.shape)
print('test set: ',x_test.shape)
```

11.1.4. Dataset Visualization

PYTHON CODE:

```
plt.figure(figsize=(20,7))
plt.suptitle('Sample Images of Dataset',fontsize=16)
for i in range(24):
    plt.subplot(3,8,i+1)
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(x_train[i].squeeze(), cmap='gray')
    plt.xlabel(str(np.argmax(y_train[i])),fontsize=12)
plt.show()
```



11.2. Building a Fully Connected Neural Network

First, we will build a fully connected neural network. Our images are 28 x 28 dimension. This means we can use a fully connected neural network and feed those pixels in a long **28 x 28 = 784** vector to our neural network. It has already been explained how to choose the number of layers or neurons per layer to use for different problems. In this model, we will use a 3-layered neural network:

PYTHON CODE:

```
def build_simple_NN_model(inputshape):
  model=Sequential()

  # To flatten the input image of 28 x 28 dimensions
  model.add(Flatten(input_shape=inputshape))

  # A dense layer of 128 neurons
  model.add(Dense(128))
  model.add(Activation('relu'))

  # A dense layer of 64 neurons
  model.add(Dense(64))
  model.add(Activation('relu'))

  # A dense layer of 10 neurons representing our 10 classes
  model.add(Dense(10))
  model.add(Activation('softmax'))

  return model
```

11.2.1. Model Summary

This will show the details of the model that we have just built here. It shows different layers of the model, their number of parameters, and the output shape of each layer. We need to provide the input shape of our images.

PYTHON CODE:

```
build_simple_NN_model(inputshape=(28,28,1)).summary()
```

```
WARNING: Logging before flag parsing goes to stderr.
W0617 15:55:42.852184 140062600333184 deprecation_wrapper.
py:119] From /usr/local/lib/python3.6/dist-packages/keras/
backend/tensorflow_backend.py:74: The name tf.get_default_graph
is deprecated. Please use tf.compat.v1.get_default_graph
instead.
```

```
W0617 15:55:42.896030 140062600333184 deprecation_wrapper.
py:119] From /usr/local/lib/python3.6/dist-packages/keras/
backend/tensorflow_backend.py:517: The name tf.placeholder is
deprecated. Please use tf.compat.v1.placeholder instead.

W0617 15:55:42.925444 140062600333184 deprecation_wrapper.
py:119] From /usr/local/lib/python3.6/dist-packages/keras/
backend/tensorflow_backend.py:4138: The name tf.random_uniform is
deprecated. Please use tf.random.uniform instead.

W0617 15:55:42.946271 140062600333184 deprecation_wrapper.
py:119] From /usr/local/lib/python3.6/dist-packages/keras/
backend/tensorflow_backend.py:133: The name tf.placeholder_with_
default is deprecated. Please use tf.compat.v1.placeholder_with_
default instead.

W0617 15:55:42.959138 140062600333184 deprecation.py:506] From /
usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_
backend.py:3445: calling dropout (from tensorflow.python.ops.
nn_ops) with keep_prob is deprecated and will be removed in a
future version.

Instructions for updating:

Please use `rate` instead of `keep_prob`. Rate should be set to
`rate = 1 - keep_prob`.
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_1 (Flatten)          (None, 784)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               100480
_________________________________________________________________
activation_1 (Activation)    (None, 128)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256
_________________________________________________________________
activation_2 (Activation)    (None, 64)                0
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                650
_________________________________________________________________
activation_3 (Activation)    (None, 10)                0
=================================================================
Total params: 109,386
Trainable params: 109,386
Non-trainable params: 0
```

11.3. Building Convolutional Neural Network

Now, we will build a convolutional neural network. We can use any predefined CNN architecture, e.g., AlexNet, LeNet, VGG, etc. But for our understanding, we will build a simple convolutional network that can learn the patterns to differentiate between the different digits. We will use the same model as we used in the binary classification and will make a change in the last layer as we need 10 neurons in the last layer to classify 10 digits.

PYTHON CODE:

```
def build_CNN_model(inputshape):
  model=Sequential()

  model.add(Conv2D(64,(3,3),input_shape=inputshape))
  model.add(Activation('relu'))
  model.add(MaxPooling2D(pool_size=2,strides=2))

  model.add(Conv2D(64,(3,3)))
  model.add(Activation('relu'))
  model.add(MaxPooling2D(pool_size=2,strides=2))


  model.add(Conv2D(32,(3,3)))
  model.add(Activation('relu'))
  model.add(MaxPooling2D(pool_size=2,strides=2))

  #this will convert all the pixels of image to flat tensor
  model.add(Flatten())

  #add some fully connected layers
  model.add(Dense(1024))
  model.add(Activation('relu'))
```

```
  #a fully connected layer with 10 neurons that represent our
10 classes
  #(one for each digit)
  model.add(Dense(10))

  #Softmax activation to get probability of each class
  model.add(Activation('softmax'))
  return model
```

11.3.1. Model Summary

PYTHON CODE:

```
build_CNN_model(inputshape=(28,28,1)).summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 26, 26, 64)        640
_________________________________________________________________
activation_17 (Activation)   (None, 26, 26, 64)        0
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 13, 13, 64)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 11, 11, 64)        36928
_________________________________________________________________
activation_18 (Activation)   (None, 11, 11, 64)        0
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 3, 3, 32)          18464
_________________________________________________________________
activation_19 (Activation)   (None, 3, 3, 32)          0
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 1, 1, 32)          0
_________________________________________________________________
flatten_5 (Flatten)          (None, 32)                0
_________________________________________________________________
dense_11 (Dense)             (None, 1024)              33792
_________________________________________________________________
activation_20 (Activation)   (None, 1024)              0
_________________________________________________________________
dense_12 (Dense)             (None, 10)                10250
_________________________________________________________________
activation_21 (Activation)   (None, 10)                0
=================================================================
Total params: 100,074
Trainable params: 100,074
Non-trainable params: 0
```

11.4. Training

We will configure some parameters for training. We will train our model for 50 epochs and with batchsize = 64.

PYTHON CODE:

```
batchsize=64
epochs=50
nnweightsdir='nncheckpoints/'
cnnweightsdir='cnncheckpoints/'
logdir='logs/'

#save model weights after every 5 epochs.
saveafter=5
create_directory(nnweightsdir)
create_directory(cnnweightsdir)
create_directory(logdir)
```

Checkpoint: The model weights will be stored after five epochs (defined by saveafter variable above). This will allow us to retrieve the model weights later, for further training or model evaluation.

PYTHON CODE:

```
batchsize=64
nn_checkpoints=keras.callbacks.ModelCheckpoint(os.path.
join(nnweightsdir,"weights.{epoch:02d}-{val_loss:.2f}.hdf5"),
monitor='val_loss', verbose=0, save_best_only=False, save_
weights_only=False, mode='auto', period=saveafter)

cnn_checkpoints=keras.callbacks.ModelCheckpoint(os.path.
join(cnnweightsdir,"weights.{epoch:02d}-{val_loss:.2f}.
hdf5"),monitor='val_loss', verbose=0, save_best_only=False,
save_weights_only=False, mode='auto', period=saveafter)

checkpoints=[nn_checkpoints,cnn_checkpoints]
```

11.4.1. Training Fully Connected Neural Network and CNN models

For multiclass classification, we will use categorical cross-entropy loss and rmsprop optimizer to train our models. In the cell below, we will train both the models (Fully Connected and Convolutional).

PYTHON CODE:

```
model_titles=['Fully Connected Neural Network','Convolutional
Neural Network']
model_outputs=[]
models=[build_simple_NN_model(inputshape=(28,28,1)),build_CNN_
model(inputshape=(28,28,1))]
for it,model in enumerate(models):
  print('Training',model_titles[it])
  model.compile(loss='categorical_crossentropy',
metrics=['accuracy'],optimizer='rmsprop')
  model_outputs.append(model.fit(x=x_train,y=y_train,
epochs=epochs,validation_data=(x_val, y_val),
  shuffle=True,callbacks=[checkpoints[it]],batch_
size=batchsize))
```

```
Training Fully Connected Neural Network
Train on 54000 samples, validate on 6000 samples
Epoch 1/50
54000/54000 [==============================] - 11s 212us/step - loss: 0.2393 - acc: 0.9303 - val_loss: 0.1473 - val_acc: 0.9573
Epoch 2/50
54000/54000 [==============================] - 9s 175us/step - loss: 0.1189 - acc: 0.9668 - val_loss: 0.1230 - val_acc: 0.9682
Epoch 3/50
54000/54000 [==============================] - 10s 179us/step - loss: 0.0955 - acc: 0.9749 - val_loss: 0.1169 - val_acc: 0.9727
Epoch 4/50
54000/54000 [==============================] - 9s 171us/step - loss: 0.0873 - acc: 0.9784 - val_loss: 0.1189 - val_acc: 0.9745
Epoch 5/50
54000/54000 [==============================] - 9s 171us/step - loss: 0.0800 - acc: 0.9807 - val_loss: 0.1360 - val_acc: 0.9725
Epoch 6/50
54000/54000 [==============================] - 9s 173us/step - loss: 0.0735 - acc: 0.9837 - val_loss: 0.1684 - val_acc: 0.9673
Epoch 7/50
54000/54000 [==============================] - 9s 172us/step - loss: 0.0667 - acc: 0.9844 - val_loss: 0.1406 - val_acc: 0.9735
Epoch 8/50
54000/54000 [==============================] - 9s 173us/step - loss: 0.0622 - acc: 0.9863 - val_loss: 0.1377 - val_acc: 0.9747
Epoch 9/50
54000/54000 [==============================] - 9s 172us/step - loss: 0.0573 - acc: 0.9875 - val_loss: 0.1465 - val_acc: 0.9745
Epoch 10/50
54000/54000 [==============================] - 9s 171us/step - loss: 0.0562 - acc: 0.9879 - val_loss: 0.1372 - val_acc: 0.9745
Epoch 11/50
54000/54000 [==============================] - 9s 171us/step - loss: 0.0521 - acc: 0.9891 - val_loss: 0.1564 - val_acc: 0.9712
Epoch 12/50
54000/54000 [==============================] - 9s 170us/step - loss: 0.0523 - acc: 0.9893 - val_loss: 0.1767 - val_acc: 0.9752
Epoch 13/50
54000/54000 [==============================] - 9s 171us/step - loss: 0.0502 - acc: 0.9900 - val_loss: 0.1603 - val_acc: 0.9768
Epoch 14/50
54000/54000 [==============================] - 10s 179us/step - loss: 0.0496 - acc: 0.9904 - val_loss: 0.1799 - val_acc: 0.9765
```

```
Epoch 15/50
54000/54000 [==============================] - 10s 187us/step - loss: 0.0446 - acc: 0.9915 - val_loss: 0.1941 - val_acc: 0.9725
Epoch 16/50
54000/54000 [==============================] - 9s 172us/step - loss: 0.0426 - acc: 0.9921 - val_loss: 0.2092 - val_acc: 0.9722
Epoch 17/50
54000/54000 [==============================] - 10s 182us/step - loss: 0.0413 - acc: 0.9925 - val_loss: 0.2029 - val_acc: 0.9747
Epoch 18/50
54000/54000 [==============================] - 9s 175us/step - loss: 0.0372 - acc: 0.9931 - val_loss: 0.1927 - val_acc: 0.9762
Epoch 19/50
54000/54000 [==============================] - 10s 179us/step - loss: 0.0378 - acc: 0.9935 - val_loss: 0.2037 - val_acc: 0.9753
Epoch 20/50
54000/54000 [==============================] - 9s 174us/step - loss: 0.0328 - acc: 0.9943 - val_loss: 0.1886 - val_acc: 0.9758
Epoch 21/50
54000/54000 [==============================] - 9s 170us/step - loss: 0.0354 - acc: 0.9938 - val_loss: 0.1923 - val_acc: 0.9762
Epoch 22/50
54000/54000 [==============================] - 9s 171us/step - loss: 0.0320 - acc: 0.9945 - val_loss: 0.2163 - val_acc: 0.9743
Epoch 23/50
54000/54000 [==============================] - 9s 174us/step - loss: 0.0310 - acc: 0.9947 - val_loss: 0.2039 - val_acc: 0.9752
Epoch 24/50
54000/54000 [==============================] - 10s 178us/step - loss: 0.0310 - acc: 0.9949 - val_loss: 0.2089 - val_acc: 0.9750
Epoch 25/50
54000/54000 [==============================] - 10s 183us/step - loss: 0.0305 - acc: 0.9951 - val_loss: 0.2304 - val_acc: 0.9752
Epoch 26/50
54000/54000 [==============================] - 10s 178us/step - loss: 0.0289 - acc: 0.9952 - val_loss: 0.2134 - val_acc: 0.9757
Epoch 27/50
54000/54000 [==============================] - 10s 179us/step - loss: 0.0272 - acc: 0.9954 - val_loss: 0.2389 - val_acc: 0.9738
Epoch 28/50
54000/54000 [==============================] - 10s 179us/step - loss: 0.0271 - acc: 0.9961 - val_loss: 0.2476 - val_acc: 0.9737
Epoch 29/50
54000/54000 [==============================] - 10s 184us/step - loss: 0.0252 - acc: 0.9961 - val_loss: 0.2407 - val_acc: 0.9743
Epoch 30/50
54000/54000 [==============================] - 10s 194us/step - loss: 0.0253 - acc: 0.9965 - val_loss: 0.2323 - val_acc: 0.9743
Epoch 31/50
54000/54000 [==============================] - 10s 185us/step - loss: 0.0247 - acc: 0.9963 - val_loss: 0.2549 - val_acc: 0.9725
Epoch 32/50
54000/54000 [==============================] - 10s 186us/step - loss: 0.0248 - acc: 0.9966 - val_loss: 0.2390 - val_acc: 0.9750
Epoch 33/50
54000/54000 [==============================] - 10s 194us/step - loss: 0.0251 - acc: 0.9965 - val_loss: 0.2360 - val_acc: 0.9758
Epoch 34/50
54000/54000 [==============================] - 11s 197us/step - loss: 0.0251 - acc: 0.9965 - val_loss: 0.2236 - val_acc: 0.9768
Epoch 35/50
54000/54000 [==============================] - 11s 198us/step - loss: 0.0230 - acc: 0.9968 - val_loss: 0.2568 - val_acc: 0.9745
Epoch 36/50
54000/54000 [==============================] - 10s 186us/step - loss: 0.0232 - acc: 0.9970 - val_loss: 0.2492 - val_acc: 0.9767
Epoch 37/50
54000/54000 [==============================] - 10s 185us/step - loss: 0.0217 - acc: 0.9970 - val_loss: 0.2310 - val_acc: 0.9763
Epoch 38/50
54000/54000 [==============================] - 10s 178us/step - loss: 0.0221 - acc: 0.9967 - val_loss: 0.2394 - val_acc: 0.9763
Epoch 39/50
54000/54000 [==============================] - 10s 190us/step - loss: 0.0232 - acc: 0.9969 - val_loss: 0.2361 - val_acc: 0.9767
Epoch 40/50
54000/54000 [==============================] - 11s 195us/step - loss: 0.0209 - acc: 0.9971 - val_loss: 0.2495 - val_acc: 0.9745
Epoch 41/50
54000/54000 [==============================] - 10s 194us/step - loss: 0.0213 - acc: 0.9973 - val_loss: 0.2620 - val_acc: 0.9748
Epoch 42/50
54000/54000 [==============================] - 11s 197us/step - loss: 0.0221 - acc: 0.9969 - val_loss: 0.2583 - val_acc: 0.9765
Epoch 43/50
54000/54000 [==============================] - 10s 192us/step - loss: 0.0218 - acc: 0.9972 - val_loss: 0.2330 - val_acc: 0.9778
Epoch 44/50
54000/54000 [==============================] - 11s 199us/step - loss: 0.0192 - acc: 0.9978 - val_loss: 0.2446 - val_acc: 0.9768
Epoch 45/50
54000/54000 [==============================] - 11s 195us/step - loss: 0.0181 - acc: 0.9977 - val_loss: 0.2472 - val_acc: 0.9770
Epoch 46/50
54000/54000 [==============================] - 11s 208us/step - loss: 0.0190 - acc: 0.9978 - val_loss: 0.2778 - val_acc: 0.9745
Epoch 47/50
54000/54000 [==============================] - 11s 197us/step - loss: 0.0194 - acc: 0.9976 - val_loss: 0.2260 - val_acc: 0.9795
Epoch 48/50
54000/54000 [==============================] - 10s 192us/step - loss: 0.0198 - acc: 0.9976 - val_loss: 0.2436 - val_acc: 0.9775
Epoch 49/50
54000/54000 [==============================] - 10s 189us/step - loss: 0.0159 - acc: 0.9981 - val_loss: 0.2694 - val_acc: 0.9757
Epoch 50/50
54000/54000 [==============================] - 10s 183us/step - loss: 0.0187 - acc: 0.9977 - val_loss: 0.2369 - val_acc: 0.9775
```

```
Training Convolutional Neural Network
Train on 54000 samples, validate on 6000 samples
Epoch 1/50
54000/54000 [==============================] - 96s 2ms/step - loss: 0.1915 - acc: 0.9400 - val_loss: 0.0826 - val_acc: 0.9748
Epoch 2/50
54000/54000 [==============================] - 98s 2ms/step - loss: 0.0787 - acc: 0.9777 - val_loss: 0.0747 - val_acc: 0.9778
Epoch 3/50
54000/54000 [==============================] - 97s 2ms/step - loss: 0.0689 - acc: 0.9814 - val_loss: 0.0701 - val_acc: 0.9820
Epoch 4/50
54000/54000 [==============================] - 96s 2ms/step - loss: 0.0692 - acc: 0.9821 - val_loss: 0.0687 - val_acc: 0.9807
Epoch 5/50
54000/54000 [==============================] - 96s 2ms/step - loss: 0.0723 - acc: 0.9836 - val_loss: 0.1193 - val_acc: 0.9798
Epoch 6/50
54000/54000 [==============================] - 96s 2ms/step - loss: 0.0729 - acc: 0.9833 - val_loss: 0.0666 - val_acc: 0.9838
Epoch 7/50
54000/54000 [==============================] - 96s 2ms/step - loss: 0.0775 - acc: 0.9841 - val_loss: 0.1084 - val_acc: 0.9797
Epoch 8/50
54000/54000 [==============================] - 96s 2ms/step - loss: 0.0759 - acc: 0.9851 - val_loss: 0.1534 - val_acc: 0.9678
Epoch 9/50
54000/54000 [==============================] - 96s 2ms/step - loss: 0.0736 - acc: 0.9854 - val_loss: 0.1696 - val_acc: 0.9732
Epoch 10/50
54000/54000 [==============================] - 96s 2ms/step - loss: 0.0773 - acc: 0.9866 - val_loss: 0.1114 - val_acc: 0.9822
Epoch 11/50
54000/54000 [==============================] - 96s 2ms/step - loss: 0.0778 - acc: 0.9860 - val_loss: 0.1344 - val_acc: 0.9812
Epoch 12/50
54000/54000 [==============================] - 95s 2ms/step - loss: 0.0785 - acc: 0.9866 - val_loss: 0.1383 - val_acc: 0.9837
Epoch 13/50
54000/54000 [==============================] - 95s 2ms/step - loss: 0.0825 - acc: 0.9876 - val_loss: 0.0825 - val_acc: 0.9823
Epoch 14/50
54000/54000 [==============================] - 95s 2ms/step - loss: 0.0925 - acc: 0.9872 - val_loss: 0.1162 - val_acc: 0.9883
Epoch 15/50
54000/54000 [==============================] - 95s 2ms/step - loss: 0.0934 - acc: 0.9881 - val_loss: 0.2590 - val_acc: 0.9748
Epoch 16/50
54000/54000 [==============================] - 95s 2ms/step - loss: 0.0784 - acc: 0.9884 - val_loss: 0.1137 - val_acc: 0.9850
Epoch 17/50
54000/54000 [==============================] - 95s 2ms/step - loss: 0.0866 - acc: 0.9893 - val_loss: 0.1314 - val_acc: 0.9840
Epoch 18/50
54000/54000 [==============================] - 96s 2ms/step - loss: 0.0910 - acc: 0.9894 - val_loss: 0.1209 - val_acc: 0.9827
Epoch 19/50
54000/54000 [==============================] - 96s 2ms/step - loss: 0.0862 - acc: 0.9901 - val_loss: 0.2244 - val_acc: 0.9833
Epoch 20/50
54000/54000 [==============================] - 96s 2ms/step - loss: 0.0998 - acc: 0.9893 - val_loss: 0.2632 - val_acc: 0.9793
Epoch 21/50
54000/54000 [==============================] - 97s 2ms/step - loss: 0.1107 - acc: 0.9903 - val_loss: 0.2219 - val_acc: 0.9828
Epoch 22/50
54000/54000 [==============================] - 97s 2ms/step - loss: 0.1125 - acc: 0.9904 - val_loss: 0.2062 - val_acc: 0.9848
Epoch 23/50
54000/54000 [==============================] - 97s 2ms/step - loss: 0.1158 - acc: 0.9903 - val_loss: 0.1789 - val_acc: 0.9853
Epoch 24/50
54000/54000 [==============================] - 97s 2ms/step - loss: 0.1223 - acc: 0.9904 - val_loss: 0.2498 - val_acc: 0.9827
Epoch 25/50
54000/54000 [==============================] - 96s 2ms/step - loss: 0.1308 - acc: 0.9900 - val_loss: 0.2993 - val_acc: 0.9788
Epoch 26/50
54000/54000 [==============================] - 98s 2ms/step - loss: 0.1238 - acc: 0.9906 - val_loss: 0.2816 - val_acc: 0.9807
Epoch 27/50
54000/54000 [==============================] - 97s 2ms/step - loss: 0.1107 - acc: 0.9915 - val_loss: 0.2439 - val_acc: 0.9818
Epoch 28/50
54000/54000 [==============================] - 97s 2ms/step - loss: 0.1190 - acc: 0.9908 - val_loss: 0.2054 - val_acc: 0.9855
Epoch 29/50
54000/54000 [==============================] - 97s 2ms/step - loss: 0.1123 - acc: 0.9917 - val_loss: 0.2352 - val_acc: 0.9832
Epoch 30/50
54000/54000 [==============================] - 97s 2ms/step - loss: 0.1372 - acc: 0.9904 - val_loss: 0.2666 - val_acc: 0.9812
Epoch 31/50
54000/54000 [==============================] - 97s 2ms/step - loss: 0.1429 - acc: 0.9901 - val_loss: 0.2226 - val_acc: 0.9852
Epoch 32/50
54000/54000 [==============================] - 97s 2ms/step - loss: 0.1346 - acc: 0.9906 - val_loss: 0.2137 - val_acc: 0.9852
Epoch 33/50
54000/54000 [==============================] - 97s 2ms/step - loss: 0.1466 - acc: 0.9901 - val_loss: 0.2080 - val_acc: 0.9858
Epoch 34/50
54000/54000 [==============================] - 97s 2ms/step - loss: 0.1356 - acc: 0.9905 - val_loss: 0.2165 - val_acc: 0.9843
Epoch 35/50
54000/54000 [==============================] - 96s 2ms/step - loss: 0.1205 - acc: 0.9915 - val_loss: 0.2775 - val_acc: 0.9815
Epoch 36/50
54000/54000 [==============================] - 96s 2ms/step - loss: 0.1341 - acc: 0.9911 - val_loss: 0.2144 - val_acc: 0.9857
Epoch 37/50
54000/54000 [==============================] - 96s 2ms/step - loss: 0.1233 - acc: 0.9915 - val_loss: 0.2678 - val_acc: 0.9822
Epoch 38/50
54000/54000 [==============================] - 96s 2ms/step - loss: 0.1457 - acc: 0.9898 - val_loss: 0.1937 - val_acc: 0.9870
Epoch 39/50
54000/54000 [==============================] - 97s 2ms/step - loss: 0.1264 - acc: 0.9915 - val_loss: 0.2041 - val_acc: 0.9867
Epoch 40/50
54000/54000 [==============================] - 97s 2ms/step - loss: 0.1264 - acc: 0.9914 - val_loss: 0.2869 - val_acc: 0.9812
Epoch 41/50
54000/54000 [==============================] - 98s 2ms/step - loss: 0.1313 - acc: 0.9911 - val_loss: 0.2174 - val_acc: 0.9855
Epoch 42/50
```

```
Epoch 42/50
54000/54000 [==============================] - 98s 2ms/step - loss: 0.1390 - acc: 0.9907 - val_loss: 0.2202 - val_acc: 0.9857
Epoch 43/50
54000/54000 [==============================] - 98s 2ms/step - loss: 0.1359 - acc: 0.9909 - val_loss: 0.2156 - val_acc: 0.9863
Epoch 44/50
54000/54000 [==============================] - 97s 2ms/step - loss: 0.1460 - acc: 0.9903 - val_loss: 0.2885 - val_acc: 0.9812
Epoch 45/50
54000/54000 [==============================] - 97s 2ms/step - loss: 0.1506 - acc: 0.9901 - val_loss: 0.2313 - val_acc: 0.9850
Epoch 46/50
54000/54000 [==============================] - 97s 2ms/step - loss: 0.1723 - acc: 0.9887 - val_loss: 0.2358 - val_acc: 0.9848
Epoch 47/50
54000/54000 [==============================] - 97s 2ms/step - loss: 0.1376 - acc: 0.9908 - val_loss: 0.2341 - val_acc: 0.9845
Epoch 48/50
54000/54000 [==============================] - 96s 2ms/step - loss: 0.1324 - acc: 0.9912 - val_loss: 0.2125 - val_acc: 0.9858
Epoch 49/50
54000/54000 [==============================] - 96s 2ms/step - loss: 0.1357 - acc: 0.9911 - val_loss: 0.2127 - val_acc: 0.9862
Epoch 50/50
54000/54000 [==============================] - 96s 2ms/step - loss: 0.1274 - acc: 0.9916 - val_loss: 0.1936 - val_acc: 0.9870
```

PYTHON CODE:

```
def summarize_stats(history,title):
  # loss
  plt.figure(figsize=(16,4))
  plt.subplot(1,2,1)
  plt.suptitle(title,fontsize=18)
  plt.title('Loss',fontsize=16)
  plt.plot(history.history['loss'], color='red',
label='Train')
  plt.plot(history.history['val_loss'], color='blue',
label='Test')
  plt.legend(loc='upper right')
  # accuracy
  plt.subplot(1,2,2)
  plt.title('Classification Accuracy',fontsize=16)
  plt.plot(history.history['acc'], color='red', label='Train')
  plt.plot(history.history['val_acc'], color='blue',
label='Test')

  plt.legend(loc='lower right')
  plt.show()

for i in range(2):
  summarize_stats(model_outputs[i],model_titles[i])
```

11.5. Testing

11.5.1. Model Weights

The weights of each model have been stored as checkpoints in the two directories viz., nncheckpoints and cnncheckpoints. We will list down those weight files here.

PYTHON CODE:

```
checkpoint_folders=['nncheckpoints','cnncheckpoints']
for f in checkpoint_folders:
  print(f,':')
  print(os.listdir(f))
```

```
nncheckpoints :
['weights.05-0.15.hdf5', 'weights.15-0.19.
hdf5', 'weights.45-0.27.hdf5', 'weights.50-0.25.
hdf5', 'weights.10-0.19.hdf5', 'weights.10-0.14.
hdf5', 'weights.40-0.25.hdf5', 'weights.05-0.14.
hdf5', 'weights.20-0.23.hdf5', 'weights.45-0.25.
hdf5', 'weights.20-0.19.hdf5', 'weights.30-0.26.
hdf5', 'weights.25-0.23.hdf5', 'weights.30-0.23.
hdf5', 'weights.50-0.24.hdf5', 'weights.15-0.20.hdf5',
'weights.40-0.28.hdf5', 'weights.35-0.26.hdf5']
cnncheckpoints :
['weights.25-0.29.hdf5', 'weights.50-0.19.
hdf5', 'weights.45-0.23.hdf5', 'weights.10-0.39.
hdf5', 'weights.20-0.26.hdf5', 'weights.15-0.26.
hdf5', 'weights.05-0.09.hdf5', 'weights.05-0.12.
hdf5', 'weights.30-0.27.hdf5', 'weights.50-0.22.
hdf5', 'weights.45-0.25.hdf5', 'weights.10-0.11.
hdf5', 'weights.35-0.28.hdf5', 'weights.25-0.30.
hdf5', 'weights.30-0.16.hdf5', 'weights.20-0.12.
hdf5', 'weights.40-0.19.hdf5', 'weights.15-0.20.hdf5',
'weights.40-0.29.hdf5', 'weights.35-0.15.hdf5']
```

11.5.2. Weight Loading

For our two models, i.e., the fully connected and convolutional neural network, we will load weights from the stored files and evaluate the models on our test data. For example, the following two lines select the last weight files in each of the directories.

PYTHON CODE:

```
#models[0] is neural network model while models[1] is
convolutional model.
#You have to select the names of weight files that you want to
load

models[0].load_weights(glob.glob('nncheckpoints/*')[-1])
models[1].load_weights(glob.glob('cnncheckpoints/*')[-1])

for it,model in enumerate(models):
  print(model_titles[it])
  test_loss, accuracy = model.evaluate(x_test,y_test,
verbose=1)
  print('Evaluation loss: %.2f, Accuracy: %.2f' % (test_
loss,accuracy * 100.0))
```

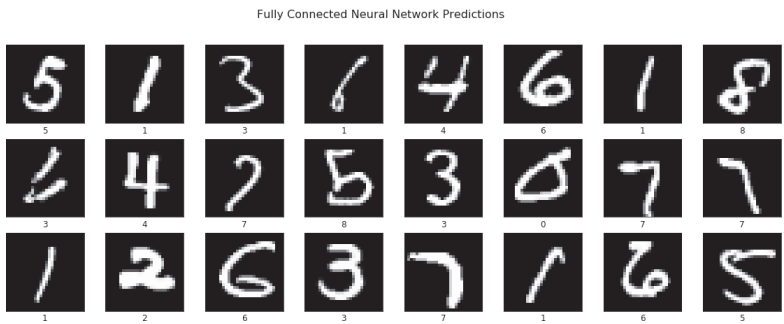
```
Fully Connected Neural Network
10000/10000 [==============================] - 0s 27us/step
Evaluation loss: 0.26, Accuracy: 97.66
Convolutional Neural Network
10000/10000 [==============================] - 4s 444us/step
Evaluation loss: 0.22, Accuracy: 98.59
```

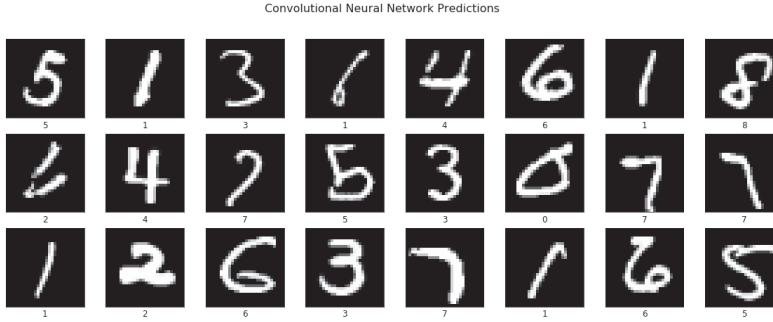
As we can see, the convolutional neural network performs a little better than a fully connected network. This is mainly because CNN is able to grasp the context and look for patterns that might not be easy in the case of fully connected (flattened pixels).

11.5.3. Visualizing Predictions

PYTHON CODE:

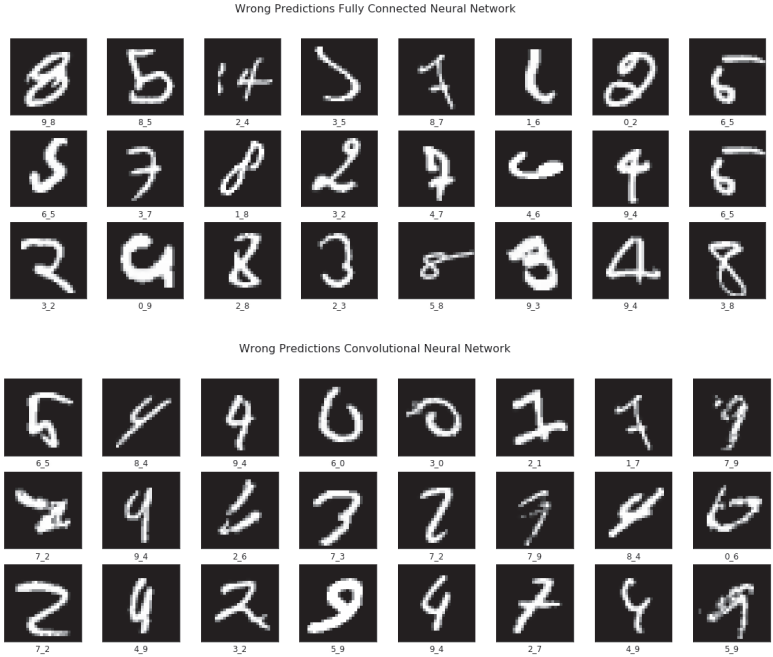
```
predictions=[]
for model in models:
  predictions.append(model.predict(x_test))
random_selection=np.random.choice(len(x_test),24)

for it,prediction in enumerate(predictions):
  plt.figure(figsize=(20,7))
  plt.suptitle(model_titles[it]+str('
Predictions'),fontsize=16)
  for i in range(24):
      plt.subplot(3,8,i+1)
      plt.grid(False)
      plt.xticks([])
      plt.yticks([])
      plt.imshow(x_test[random_selection[i]].squeeze(),
cmap='gray')
      plt.xlabel(str(np.argmax(prediction[random_
selection[i]])),fontsize=12)
  plt.show()
```





PYTHON CODE:

```
for it,prediction in enumerate(predictions):
  predicted=np.argmax(prediction,axis=1)
  actual=np.argmax(y_test,axis=1)
  indices=list(np.where(predicted!=actual))[0]

  random_selection=np.random.choice(len(indices),24)
  plt.figure(figsize=(20,7))
  plt.suptitle('Wrong Predictions '+str(model_
titles[it]),fontsize=16)
  for i in range(24):
      plt.subplot(3,8,i+1)
      plt.grid(False)
      plt.xticks([])
      plt.yticks([])
      plt.imshow(x_test[indices[random_selection[i]]].
squeeze(), cmap='gray')
      plt.xlabel(str(np.argmax(prediction[indices[random_
selection[i]]]))+str('_')+str(np.argmax(y_test[indices[random_
selection[i]]])),fontsize=12)
  plt.show()
```



11.5.4. Confusion Matrix

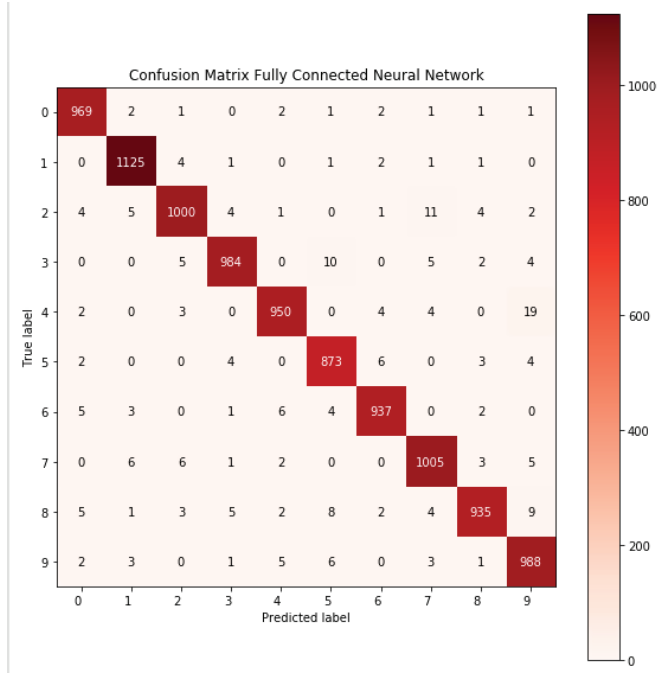
Confusion matrix evaluates the model to see which classes have been correctly classified most often and which classes are hard for the model to classify. The diagonal elements show for which the actual label of the image and model predicted label are the same, while the off-diagonal elements show misclassified images.

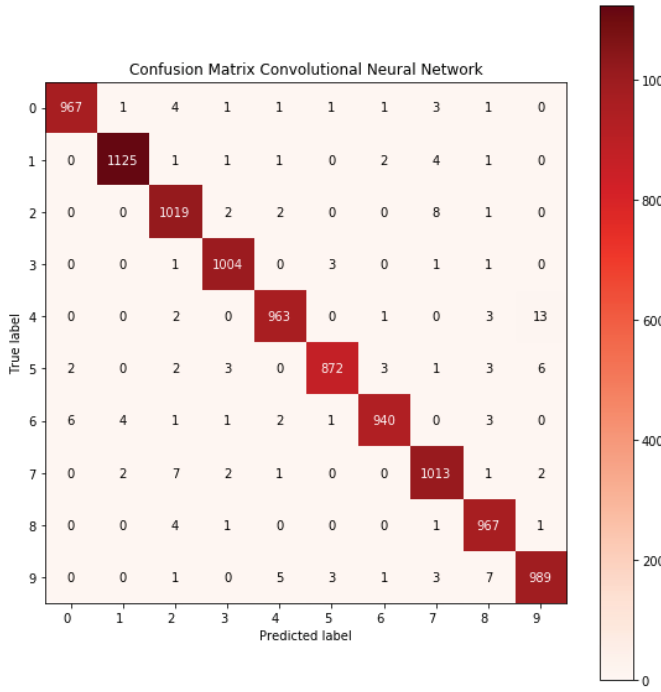
PYTHON CODE:

```
def plot_confusion_matrix(y_true, y_pred, titletxt,
                          cmap=plt.cm.Reds):

    cm = confusion_matrix(y_true, y_pred)
    classes = range(0,10)
```

```
    fig, ax = plt.subplots(figsize=(8,8))
    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
    # We want to show all ticks...
    ax.set(xticks=np.arange(cm.shape[1]),
           yticks=np.arange(cm.shape[0]),
           # ... and label them with the respective list
entries
           xticklabels=classes, yticklabels=classes,
           title='Confusion Matrix '+str(titletxt),
           ylabel='True label',
           xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), ha="right",
             rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
    fmt = 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else
"black")
    fig.tight_layout()
    return ax


for it,prediction in enumerate(predictions):
  predicted=np.argmax(prediction,axis=1)
  actual=np.argmax(y_test,axis=1)
  plot_confusion_matrix(actual,predicted,model_titles[it])
  print(model_titles[it],': \nAccuracy Metrics for each
digit:')
  print(classification_report(actual,predicted))
```

```
Fully Connected Neural Network :
Accuracy Metrics for each digit:
              precision    recall  f1-score   support

           0       0.98      0.99      0.98       980
           1       0.98      0.99      0.99      1135
           2       0.98      0.97      0.97      1032
           3       0.98      0.97      0.98      1010
           4       0.98      0.97      0.97       982
           5       0.97      0.98      0.97       892
           6       0.98      0.98      0.98       958
           7       0.97      0.98      0.97      1028
           8       0.98      0.96      0.97       974
           9       0.96      0.98      0.97      1009

    accuracy                           0.98     10000
   macro avg       0.98      0.98      0.98     10000
weighted avg       0.98      0.98      0.98     10000

Convolutional Neural Network :
Accuracy Metrics for each digit:
              precision    recall  f1-score   support


Convolutional Neural Network :
Accuracy Metrics for each digit:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       980
           1       0.99      0.99      0.99      1135
           2       0.98      0.99      0.98      1032
           3       0.99      0.99      0.99      1010
           4       0.99      0.98      0.98       982
           5       0.99      0.98      0.98       892
           6       0.99      0.98      0.99       958
           7       0.98      0.99      0.98      1028
           8       0.98      0.99      0.99       974
           9       0.98      0.98      0.98      1009

    accuracy                           0.99     10000
   macro avg       0.99      0.99      0.99     10000
weighted avg       0.99      0.99      0.99     10000
```


Confusion Matrix Fully Connected Neural Network
True label
Predicted label
0 969 2 1 0 2 1 2 1 1 1
1 0 1125 4 1 0 1 2 1 1 0
2 4 5 1000 4 1 0 1 11 4 2
3 0 0 5 984 0 10 0 5 2 4
4 2 0 3 0 950 0 4 4 0 19
5 2 0 0 4 0 873 6 0 3 4
6 5 3 0 1 6 4 937 0 2 0
7 0 6 6 1 2 0 0 1005 3 5
8 5 1 3 5 2 8 2 4 935 9
9 2 3 0 1 5 6 0 3 1 988
0 1 2 3 4 5 6 7 8 9
1000
800
600
400
200
0


Confusion Matrix Convolutional Neural Network
True label
Predicted label
0 967 1 4 1 1 1 1 3 1 0
1 0 1125 1 1 1 0 2 4 1 0
2 0 0 1019 2 2 0 0 8 1 0
3 0 0 1 1004 0 3 0 1 1 0
4 0 0 2 0 963 0 1 0 3 13
5 2 0 2 3 0 872 3 1 3 6
6 6 4 1 1 2 1 940 0 3 0
7 0 2 7 2 1 0 0 1013 1 2
8 0 0 4 1 0 0 0 1 967 1
9 0 0 1 0 5 3 1 3 7 989
0 1 2 3 4 5 6 7 8 9
1000
800
600
400
200
0

12

Project 6: Fine Tuning Pretrained Model — VGG19

As we use the fine-tuned VGG16 model for our binary class classification, we will use the VGG19 for multiclass classification. As we know, the four steps to fine-tune a model are:

1. Load the pretrained model and set include_top=False to remove the last fully connected layers.
2. Freeze the initial layers because we do not want to train them.
3. Build our own model on top of the pretrained one.
4. Train the model.

We will fine-tune VGG19 model for our problem. One important point to notice is that VGG19 takes an image with three channels and should be at least 32 x 32 (height x width). Since our images are 28 x 28 x 1, we need to make some modifications to our input data.

1. To solve the height and width problem, we can simply resize the images using np.resize.

2. Convert a 1-channel image to a 3-channel image. We can use the same one channel and replicate it in depth to make a 3-dimension image.

12.1. Data Preparation

PYTHON CODE:

```
import cv2

def resize_and_add_channels(inputtensor):
  # First we will resize all images to 32 x 32
  newx_train=np.empty(shape=(len(inputtensor),32,32,1))
  for i in range(len(inputtensor)):
    newx_train[i,:,:,:]=np.expand_dims(cv2.resize(x_
train[i,:,:,:],(32,32)),axis=2)

  # Now we will make one channel image to 3 channel image
  three_channel_imgs=np.tile(newx_train,(1,1,1,3))
  return three_channel_imgs

newx_train=resize_and_add_channels(x_train)
newx_test=resize_and_add_channels(x_test)
newx_val=resize_and_add_channels(x_val)
print('New X_train: ',newx_train.shape)
print('New x_val: ',newx_val.shape)
print('New X_test: ',newx_test.shape)
```

```
New X_train:  (54000, 32, 32, 3)
New x_val:  (6000, 32, 32, 3)
New X_test:  (10000, 32, 32, 3)
```

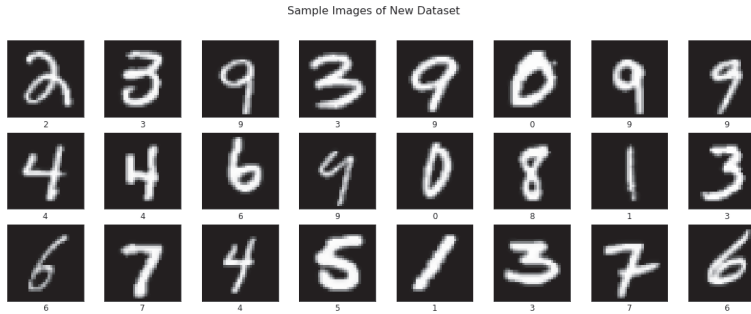
Now we have data in the form we want. We do not need to change anything from y_train, y_val, or y_test because these are one-hot encoded vectors and do not need to be changed.

12.1.1. Data Visualization

We can quickly visualize our data to verify that our images are in the correct form for training.

PYTHON CODE:

```
plt.figure(figsize=(20,7))
plt.suptitle('Sample Images of New Dataset',fontsize=16)
for i in range(24):
    plt.subplot(3,8,i+1)
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(newx_train[i].squeeze(), cmap='gray')
    plt.xlabel(str(np.argmax(y_train[i])),fontsize=12)
plt.show()
```



12.2. Loading Pretrained Model

We will remove the top three fully connected layers from VGG19 and will build our own model to classify 10 classes.

PYTHON CODE:

```
from keras.applications import VGG19
vgg19=VGG19(weights='imagenet',input_shape=(32,32,3),include_
top=False)
vgg19.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_5 (InputLayer)         (None, 32, 32, 3)         0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 32, 32, 64)        1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 32, 32, 64)        36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 16, 16, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 16, 16, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 16, 16, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 8, 8, 128)         0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 8, 8, 256)         295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 8, 8, 256)         590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 8, 8, 256)         590080
_________________________________________________________________
block3_conv4 (Conv2D)        (None, 8, 8, 256)         590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 4, 4, 256)         0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 4, 4, 512)         1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block4_conv4 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 2, 2, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 2, 2, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 2, 2, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 2, 2, 512)         2359808
_________________________________________________________________
block5_conv4 (Conv2D)        (None, 2, 2, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 1, 1, 512)         0
=================================================================
Total params: 20,024,384
Trainable params: 20,024,384
Non-trainable params: 0
_________________________________________________________________
```

As we can see, VGG19 has more Convolutional layers in each block than VGG16.

12.3. Freezing the Initial Layers

The VGG19 model has five blocks, and each block has Convolutional and Max-pooling layers. We will freeze the first four blocks, and we will train the last block to learn the features relevant to solve our classification problem. The last block contains four convolutional layers and one pool layer.

PYTHON CODE:

```
for layer in vgg19.layers[:-5]:
  layer.trainable=False

for layer in vgg19.layers:
  print(layer,layer.trainable)
```

```
<keras.engine.input_layer.InputLayer object at 0x7f4d08216ef0> False
<keras.layers.convolutional.Conv2D object at 0x7f4d08216f98> False
<keras.layers.convolutional.Conv2D object at 0x7f4d08742eb8> False
<keras.layers.pooling.MaxPooling2D object at 0x7f4d0821a1d0> False
<keras.layers.convolutional.Conv2D object at 0x7f4d135fc748> False
<keras.layers.convolutional.Conv2D object at 0x7f4d13736208> False
<keras.layers.pooling.MaxPooling2D object at 0x7f4d13751a20> False
<keras.layers.convolutional.Conv2D object at 0x7f4d13751e80> False
<keras.layers.convolutional.Conv2D object at 0x7f4d137625f8> False
<keras.layers.convolutional.Conv2D object at 0x7f4d1388a860> False
<keras.layers.convolutional.Conv2D object at 0x7f4d138a9240> False
<keras.layers.pooling.MaxPooling2D object at 0x7f4d136c7b00> False
<keras.layers.convolutional.Conv2D object at 0x7f4d136c7160> False
<keras.layers.convolutional.Conv2D object at 0x7f4d136fc630> False
<keras.layers.convolutional.Conv2D object at 0x7f4d137e29b0> False
<keras.layers.convolutional.Conv2D object at 0x7f4d137cae80> False
<keras.layers.pooling.MaxPooling2D object at 0x7f4d1387dc18> False
<keras.layers.convolutional.Conv2D object at 0x7f4d1385ee10> True
<keras.layers.convolutional.Conv2D object at 0x7f4d1387b208> True
<keras.layers.convolutional.Conv2D object at 0x7f4d1380da20> True
<keras.layers.convolutional.Conv2D object at 0x7f4d13822e48> True
<keras.layers.pooling.MaxPooling2D object at 0x7f4d13794d30> True
```

12.4. Build a Model on Top of Base Model

VGG19 model will act as our base model. Now, we will add our classifier on top of this base model. We will add the same last layers from our previous model for multiclass classification.

PYTHON CODE:

```
from keras.models import Sequential
from keras.layers.core import Dense, Activation, Flatten
from keras.layers.convolutional import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dropout

def build_model_finetune(basemodel):
  model=Sequential()

  # Add base model
  model.add(basemodel)

  # To Flatten the pixel values of base model output tensor
  model.add(Flatten())

  #add some fully connected layers
  model.add(Dense(1024))
  model.add(Activation('relu'))
  model.add(Dropout(0.25))

  model.add(Dense(512))
  model.add(Activation('relu'))
  model.add(Dropout(0.25))

  # A fully connected layer with 10 neurons for 10 classes
  model.add(Dense(10))

  # Softmax to get probabilities of each class
  model.add(Activation('softmax'))

  return model

build_model_finetune(vgg19).summary()
```

```
[ ]  build_model_finetune(vgg19).summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg19 (Model)                (None, 1, 1, 512)         20024384
_________________________________________________________________
flatten_11 (Flatten)         (None, 512)               0
_________________________________________________________________
dense_29 (Dense)             (None, 1024)              525312
_________________________________________________________________
activation_29 (Activation)   (None, 1024)              0
_________________________________________________________________
dropout_19 (Dropout)         (None, 1024)              0
_________________________________________________________________
dense_30 (Dense)             (None, 512)               524800
_________________________________________________________________
activation_30 (Activation)   (None, 512)               0
_________________________________________________________________
dropout_20 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_31 (Dense)             (None, 10)                5130
_________________________________________________________________
activation_31 (Activation)   (None, 10)                0
=================================================================
Total params: 21,079,626
Trainable params: 10,494,474
Non-trainable params: 10,585,152
_________________________________________________________________
```

12.5. Training the Model

As we trained our earlier model on two data iterators, viz., with augmentation and without augmentation, we will do the same for this model, too.

PYTHON CODE:

```
batchsize=32
epochs=10
nonaugweightsdir_ft='vgg19_nonaug_checkpoints/'
augweightsdir_ft='vgg19_aug_checkpoints/'
logdir='logs/'
categories=['Fine Tune Without Augmentation','Fine Tune With
Augmentation']
saveafter=2
```

Checkpoint: The model weights will be stored after n epochs (n is defined by saveafter variable above). This will allow us to retrieve the model weights later, for further training or model evaluation.

PYTHON CODE:

```
nonaugcheckpoints=keras.callbacks.ModelCheckpoint(os.path.
join(nonaugweightsdir_ft,"weights.{epoch:02d}-{val_loss:.2f}.
hdf5"),
  monitor='val_loss', verbose=0, save_best_only=False,
  save_weights_only=False, mode='auto', period=saveafter)
augcheckpoints=keras.callbacks.ModelCheckpoint(os.path.
join(augweightsdir_ft,"weights.{epoch:02d}-{val_loss:.2f}.
hdf5"),
  monitor='val_loss', verbose=0, save_best_only=False,
  save_weights_only=False, mode='auto', period=saveafter)
checkpoints=[nonaugcheckpoints,augcheckpoints]

from keras.optimizers import RMSprop

models=[build_model_finetune(vgg19),build_model_finetune(vgg19)]
model_outputs=[]
for it,model in enumerate(models):
  print('\nTraining',categories[it])
  model.compile(loss='categorical_
crossentropy',metrics=['accuracy'],optimizer=RMSprop(lr=1e-6))
  model_outputs.append(model.fit(x=newx_train,y=y_train,
epochs=epochs,validation_data=(newx_val, y_val),
  shuffle=True,callbacks=[checkpoints[it]],batch_
size=batchsize))
```

12.5.1. Visualizing Training Loss and Accuracies

PYTHON CODE:

```
def summarize_stats(history,title):
  # loss
  plt.figure(figsize=(18,4))
  plt.subplot(1,2,1)
  plt.suptitle(title,fontsize=18)
  plt.title('Loss',fontsize=16)
  plt.plot(history.history['loss'], color='red',
label='Train')
  plt.plot(history.history['val_loss'], color='blue',
label='Validation')
  plt.legend(loc='upper right')
  # accuracy
  plt.subplot(1,2,2)
  plt.title('Classification Accuracy',fontsize=16)
  plt.plot(history.history['acc'], color='red', label='Train')
  plt.plot(history.history['val_acc'], color='blue',
label='Validation')

  plt.legend(loc='lower right')
  plt.show()

for it,model in enumerate(models):
  summarize_stats(model_outputs[it],title=str(categories[it]))
```

12.6. Testing Pretrained Model

12.6.1. Model Weights

The weights of the model were stored in the vgg_nonaug_checkpoints/directory and the vgg_aug_checkpoints/directory, after every two epochs. We will list down those weights files.

PYTHON CODE:

```
print('\nModel for Non Augmented Data: ')
print(os.listdir(nonaugweightsdir_ft))
print('\nModel for Augmented Data: ')
print(os.listdir(augweightsdir_ft))
```

```
Model for Non Augmented Data:

['weights.06-0.59.hdf5', 'weights.02-0.64.
hdf5', 'weights.03-0.63.hdf5', 'weights.01-0.65.
hdf5', 'weights.07-0.57.hdf5', 'weights.05-0.60.
hdf5', 'weights.09-0.54.hdf5', 'weights.04-0.62.hdf5',
'weights.08-0.55.hdf5', 'weights.10-0.52.hdf5']


Model for Augmented Data:

['weights.03-0.61.hdf5', 'weights.04-0.63.
hdf5', 'weights.07-0.61.hdf5', 'weights.06-0.61.
hdf5', 'weights.08-0.58.hdf5', 'weights.09-0.53.
hdf5', 'weights.01-0.65.hdf5', 'weights.05-0.64.hdf5',
'weights.02-0.67.hdf5', 'weights.10-0.63.hdf5']
```

12.6.2. Weight Loading

For our convolutional model, we will load weights from the file and evalute on test data. We can load the weight file stored in the directory vgg_nonaug_checkpoints/ or vgg_aug_checkpoints/

PYTHON CODE:

```
models[0].load_weights(glob.glob('vgg_nonaug_checkpoints/*')
[-1])

models[1].load_weights(glob.glob('vgg_aug_checkpoints/*')[-2])

for it,model in enumerate(models):
  print(categories[it])
  test_loss, accuracy = model.evaluate_generator(testing_
iterators[it],len(testing_iterators[it]), verbose=1)

  print('Testing loss: %.2f, Accuracy: %.2f' % (test_
loss,accuracy * 100.0))
```

```
Fine Tune Without Augmentation
4/4 [==============================] - 51s 13s/step
Testing loss: 0.58, Accuracy: 71.00
Fine Tune With Augmentation
4/4 [==============================] - 55s 14s/step
Testing loss: 0.71, Accuracy: 49.00
```

12.6.3. Visualizing Predictions

PYTHON CODE:

```
predictions=[]
for it,model in enumerate(models):
  print(categories[it])
  predictions.append(model.predict_generator(testing_
iterators[it],len(testing_iterators[it])))

random_selection=np.random.choice(len(x_test),24)

for it,prediction in enumerate(predictions):
  plt.figure(figsize=(20,7))
  plt.suptitle(str(categories[it])+' - CNN Model
Predictions:',fontsize=16)
  for i in range(24):
      plt.subplot(3,8,i+1)
```

```
      plt.grid(False)
      plt.xticks([])
      plt.yticks([])
      plt.imshow(cv2.imread(x_test[random_selection[i]]),
cmap='gray')
      plt.xlabel(str(C2[np.argmax(prediction[random_
selection[i]])]),fontsize=12)
  plt.show()
```

```
Fine Tune Without Augmentation
Fine Tune With Augmentation
```

12.6.4. Confusion Matrix

Just as we used the confusion matrix to see the result stats in binary classification, CM will provide stats on the correctly and incorrectly classified instances for our 10 classes.

PYTHON CODE:

```
def plot_confusion_matrix(y_true, y_pred, titletxt,
                          cmap=plt.cm.Reds):

    cm = confusion_matrix(y_true, y_pred)
    classes = range(0,2)

    fig, ax = plt.subplots(figsize=(4,4))
    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
    # We want to show all ticks...
    ax.set(xticks=np.arange(cm.shape[1]),
           yticks=np.arange(cm.shape[0]),
           # ... and label them with the respective list entries
           xticklabels=classes, yticklabels=classes,
           title='Confusion Matrix ',
           ylabel='True label',
           xlabel='Predicted label')
```

```
    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), ha="right",
             rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
    fmt = 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else
"black")
    fig.tight_layout()
    return ax


one_hot_y_test=to_categorical(y_test,num_classes=10)
print(one_hot_y_test.shape)
for it,prediction in enumerate(predictions):
  predicted=np.argmax(prediction,axis=1)
  actual=np.argmax(np.array(one_hot_y_test),axis=1)
  plot_confusion_matrix(actual,predicted,categories[it])
  print('\n',categories[it],'Accuracy Metrics for each class:')
  print(classification_report(actual,predicted))
```

13

Project 7: Deep Learning for Text and Sequences

13.1. What is Natural Language Processing?

NLP falls under the domain of AI, which involves working with natural languages to perform various tasks like Language Translation, Speech Recognition (translate spoken language into text by computers), Spam Classification, and many more.

We have divided this chapter into two main sections:

- In the first section, we will give a brief introduction to basic NLP concepts using NLTK (Natual Language ToolKit).
- In the second section, we will go through RNN (Recurrent Neural Network) and build a model in Keras to perform semantic analysis.

13.1.1. Part I: Introduction to Basic NLP Concepts

We will be covering the following concepts in this part:

1. Tokenization (Sentence and Word Tokenization)

2. Lemmatization and Stemming
3. Stop Words Removal
4. Regex

13.2. Introduction to NLTK

Before we jump into code, let's first see what NLTK is. It is an open-source platform for working on human language data in python. It provides easily usable interfaces to text handling and manipulation programs like tokenization, stemming, lemmatization, and many more. It has a variety of corpora available to work with. In this chapter, we will be taking advantage of these libraries to demonstrate the theoretical concepts in code.

PYTHON CODE:

```
#Lets install NLTK first
!pip install --user -U nltk
```

PYTHON CODE:

```
#imports
import nltk
print ('nltk package imported successfully')
nltk.download('popular') #download some particular datasets
and models
```

```
nltk package imported successfully
[nltk_data] Downloading collection 'popular'
[nltk_data]    |
[nltk_data]    | Downloading package cmudict to /root/nltk_data...
[nltk_data]    |   Package cmudict is already up-to-date!
[nltk_data]    | Downloading package gazetteers to /root/nltk_data...
[nltk_data]    |   Package gazetteers is already up-to-date!
[nltk_data]    | Downloading package genesis to /root/nltk_data...
[nltk_data]    |   Package genesis is already up-to-date!
[nltk_data]    | Downloading package gutenberg to /root/nltk_data...
[nltk_data]    |   Package gutenberg is already up-to-date!
[nltk_data]    | Downloading package inaugural to /root/nltk_data...
[nltk_data]    |   Package inaugural is already up-to-date!
[nltk_data]    | Downloading package movie_reviews to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Package movie_reviews is already up-to-date!
[nltk_data]    | Downloading package names to /root/nltk_data...
[nltk_data]    |   Package names is already up-to-date!
[nltk_data]    | Downloading package shakespeare to /root/nltk_data...
[nltk_data]    |   Package shakespeare is already up-to-date!
[nltk_data]    | Downloading package stopwords to /root/nltk_data...
[nltk_data]    |   Package stopwords is already up-to-date!
[nltk_data]    | Downloading package treebank to /root/nltk_data...
[nltk_data]    |   Package treebank is already up-to-date!
[nltk_data]    | Downloading package twitter_samples to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Package twitter_samples is already up-to-date!
[nltk_data]    | Downloading package omw to /root/nltk_data...
[nltk_data]    |   Package omw is already up-to-date!
[nltk_data]    | Downloading package wordnet to /root/nltk_data...
[nltk_data]    |   Package wordnet is already up-to-date!
[nltk_data]    | Downloading package wordnet_ic to /root/nltk_data...
[nltk_data]    |   Package wordnet_ic is already up-to-date!
[nltk_data]    | Downloading package words to /root/nltk_data...
[nltk_data]    |   Package words is already up-to-date!
[nltk_data]    | Downloading package maxent_ne_chunker to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Package maxent_ne_chunker is already up-to-date!
[nltk_data]    | Downloading package punkt to /root/nltk_data...
[nltk_data]    |   Package punkt is already up-to-date!
[nltk_data]    | Downloading package snowball_data to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Package snowball_data is already up-to-date!
[nltk_data]    | Downloading package averaged_perceptron_tagger to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Package averaged_perceptron_tagger is already up-
[nltk_data]    |       to-date!
[nltk_data]    |
[nltk_data]  Done downloading collection popular
True
```

13.3. Tokenization

Tokenization is the process of splitting a string of language into sentences and words. This splitting is done by separating tokens (sentences and words) on the basis of punctuations. For example, sentences are split using periods and words by using blank spaces. However, it is a very simple way of defining tokenization, as this is not always true for other languages.

PYTHON CODE:

```
#Example for tokenization

paragraph = 'Representation learning has been a well known
study in the computational intelligence space for years but
its '\
+ 'importance has taken off lately with the birth of deep
learning. While conventional computational intelligence
techniques '\
+ 'such as classification often look at mathematically well-
formed datasets, deep learning models look at data such as
images '\
+ 'that have not well-structured features. In that sense,
representation learning is a an important feature of most deep
learning architectures.'

print ('Paragraph : ' , paragraph)
```

```
Paragraph:  Representation learning has been a well-known study
in the computational intelligence space for years, but its
importance has taken off lately with the birth of deep learning.
While conventional computational intelligence techniques such as
classification often look at mathematically well-formed datasets,
deep learning models look at data such as images that don't have
well-structured features. In that sense, representation learning is
an important feature of most deep learning architectures.
```

PYTHON CODE:

```
#Tokenize the paragraph into sentences
sentences = nltk.sent_tokenize(paragraph)

#Lets see the sentences
for sentence in enumerate (sentences):
  print (sentence)
```

```
(0, 'Representation learning has been a well-known study in the computational intelligence space for years, but its importance has taken off lately with the birth of deep learning.')

(1, 'While conventional computational intelligence techniques such as classification often look at mathematically well-formed datasets, deep learning models look at data such as images that don't have well-structured features.')

(2, 'In that sense, representation learning is an important feature of most deep learning architectures.')
```

```
sent_tokenize (text, language='english')
```

1. Params — This built-in nltk function takes two parameters: *text* — the string to be tokenized and *language* — the language of the string to be tokenized, which is by default set to English.
2. Working — This function uses an unsupervised algorithm to create sentences considering a whole lot of factors like words used at the start of the sentences, sentence boundaries, abbreviation words, and collocations.
3. Return — returns a list of tokenized sentences from the text.

PYTHON CODE:

```
#Now lets tokenize sentences into words
for sentence in sentences:
  #calls the function to tokenize sentence into words
  words = nltk.word_tokenize(sentence)
  #lets just print length and not all the words to avoid
clutter
  print (len(words))
```

```
29
30
17
```

```
word_tokenize(text, language='english', preserve_line=False)
```

- Params — This built-in nltk function takes three params: *text* — the string to be tokenized, *language* — language of the string to be tokenized, which is by default set to English, and *preserve_line* — flag to tokenize the text into sentences before word tokenizing it. It is false by default.
- Working — This function uses regular expressions to tokenize the text. Roughly, it standardizes the contractions, treats most punctuation token as separators, and filter periods separate that appear at the end of the sentence.
- Return — returns a list of tokenized words from the text.

13.4. Lemmatization and Stemming

Lemmatization and Stemming are used to reduce the set of derived words to a common base word.

For example, dog, dogs, dog's, and dogs' should all be reduced to one common form **dog.**

Stemming does this by chopping off the beginning or end of words to remove affixes (prefixes and suffixes). For example, *driving* is converted to *drive*.

Lemmatization does a similar thing by reducing words to their dictionary form. For example, words like best and excellent are changed to good as standard, past, future, and continuous forms of tenses are resolved to single standard form.

PYTHON CODE:

```
#Lets see the code
from nltk.stem import PorterStemmer, WordNetLemmatizer
#for providing part of speech to Lemmatizer
from nltk.corpus import wordnet

lemmatizer = WordNetLemmatizer()
stemmer = PorterStemmer()

print ('Stemmer for word drawing : ', stemmer.stem('drawing'))
print ('Stemmer for word drove : ', stemmer.stem('drove'))

print ('Lemmatizer for word drove : ', lemmatizer.
lemmatize('drove', wordnet.VERB))
print ('Lemmatizer for word drove : ', lemmatizer.
lemmatize('drawing', wordnet.VERB))
```

```
Stemmer for word drawing :  draw
Stemmer for word drove :  drove
Lemmatizer for word drove :  drive
Lemmatizer for word drove :  draw
```

`stemmer.stem(token)`

- Params — This function takes one param: token — the token to be stemmed.
- Working — This function performs common stemming functionality like removing prefixes and suffixes, converting feminine to the masculine form, and converting plural form to its singular form.

- Return: return the stemmed token.

```
lemmatizer.lemmatize(word, pos)
```

- Params — This function takes two params: *word* — the word to be converted to base form, and *pos* — it tells about the part of speech to which the word belongs.
- Working — This function performs a set of morphological operations to find the base word for the passed word in wordnet (one of the corpora available in nltk) database.
- Return: return the lemmatized word.

13.5. Stop Words Removal

Stop words are the words that most commonly used that do not add much of semantic meaning to the text. Therefore, we filter out these words before applying the algorithms to remove noise. Some common stop words include we, a, you.

PYTHON CODE:

```
#lets see the collection of stopwords in nltk
#import stopwords from nltk corpus
from nltk.corpus import stopwords

#Stopwords in English Language
print(stopwords.words("english"))
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves',
'you', "you're", "you've", "you'll", "you'd", 'your', 'yours',
'yourself', 'yourselves', 'he', 'him', 'his', 'himself',
'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its',
'itself', 'they', 'them', 'their', 'theirs', 'themselves',
'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be',
'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does',
'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or',
'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for',
```

```
'with', 'about', 'against', 'between', 'into', 'through',
'during', 'before', 'after', 'above', 'below', 'to', 'from',
'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under',
'again', 'further', 'then', 'once', 'here', 'there', 'when',
'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few',
'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not',
'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't',
'can', 'will', 'just', 'don', "don't", 'should', "should've",
'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren',
"aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn',
"doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven',
"haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn',
"mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn',
"shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won',
"won't", 'wouldn', "wouldn't"]
```

PYTHON CODE:

```
#lets see how our paragraph looks like without stopwords

stop_words = stopwords.words("english")
words = nltk.word_tokenize(paragraph)
for word in words:
  if not word in stop_words:
    print (word, sep=' ' , end=' ')
```

```
Representation learning well-known study computational
intelligence space years importance taken lately birth deep
learning. While conventional computational intelligence
techniques classification often look mathematically well-formed
datasets, deep learning models look data images well-structured
features. In sense, representation learning important feature
deep learning architectures.
```

13.6. Regex

Regex (Regular Expression) is a sequence of characters that defines a search pattern, or in simpler words, it is a method for a programmer to tell the program what patterns to look for and what actions to take once those patterns are found in the

text. Regex is a very widely used technique to cleanse data with a particular type of noise.

PYTHON CODE:

```
#lets see how to apply a simple regex in python
import re

#replace blank space with '_' in the paragraph
result = re.sub("\ ", "_", paragraph)
print(result)
```

```
Representation_learning_has_been_a_well_known_study_in_the_
computational_intelligence_space_for_years_but_its_importance_
has_taken_off_lately_with_the_birth_of_deep_learning._While_
conventional_computational_intelligence_techniques_such_as_
classification_often_look_at_mathematically_well-formed_datasets,_
deep_learning_models_look_at_data_such_as_images_that_have_not_
well-structured_features._In_that_sense,_representation_learning_
is_a_an_important_feature_of_most_deep_learning_architectures.
```

```
re.sub (pattern, repl, string, count=0)
```

- Params — This function takes four params: *pattern* — the pattern that is to replace, *repl* — replacement for the pattern, *string* — the string that is to be replaced, and *count* — the count for the maximum number of occurences that are to be replaced.
- Working: Returns an updated string, and if no matching pattern is found, then returns the unchanged string.
- Updated string after replacements (if a pattern is found).

13.6.2. Part II: Building RNN to Classify Sentiments

In this section, we will go through a basic definition of RNN (Recurrent Neural Network), LSTM (Long Short-Term Memory) Network, and build a classifier with LSTM layer in Keras to classify sentiments.

So, What is an RNN?

RNNs are artificial neural networks that, unlike other ANNs (Artificial Neural Networks), remember history from previous steps. They contain a hidden layer that is responsible for maintaining this history. These networks are particularly useful when dealing with time-series data, text sequences, or sound data.

However, RNNs are not particularly useful in solving problems that require remembering long-term sequences due to the **vanishing gradients** problem. A variation of RNNs known as LSTM is used, which contains a memory cell along with a gating mechanism to decide entry and exit for memory with the cell.

13.7. Word Embeddings

Before we jump into code, let's first discuss word embeddings. Word embeddings is a way of representing text such that the words with similar contexts have a smaller distance compared to words with different meanings. For example, words like good and best will lie closer to each other in the coordinate system.

Two famous word embedding models are:

1. Word2Vec
2. Glove

PYTHON CODE:

```
#for LA operations
import numpy as np
#for data manipulation
import pandas as pd
#for os related functions
import os

#Imports from keras and tokenizer API
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import LSTM
from keras.layers.embeddings import Embedding
from keras.preprocessing.text import Tokenizer
from keras.preprocessing import sequence
```

```
Using TensorFlow backend.
```

13.8. Model Building and Training

Now, we are all set to build our model. We will be building models—one with an LSTM layer and another with the Conv1D layer. We will be using a preloaded dataset for the LSTM network, whereas for CNN, we will be using a manually downloaded dataset. Both the datasets are the same, but we wanted to give an idea of how we can do it in two different ways.

13.9. Dataset Loading

First, we will be using the Keras preloaded dataset of a **Large movie review dataset** to give you an overview of how to use built-in datasets in Keras.

Dataset — Large Movie Dataset provides 25,000 movie reviews for training and the same number of reviews for test training,

where each review has either a positive (**representated as 1**) or negative (**represented as 0**) sentiment.

max_words — Shows the maximum length of a review. If any review is greater than that, we will truncate it and zero pad the smaller ones.

top_words — Defines how many top words to load from the dataset and zero out the words other than those top 5,000.

PYTHON CODE:

```
#we will be using preloaded dataset for LSTM and manually
prepared for Conv
from keras.datasets import imdb

#Lets first build LSTM with using imdb API for the dataset
# fix random seed for reproducibility
seed = 7
np.random.seed(seed)

np_load_old = np.load

# modify the default parameters of np.load
np.load = lambda *a,**k: np_load_old(*a, allow_pickle=True,
**k)


max_words = 500
top_words = 5000
EMBEDDING_DIM = 100

(X_train, y_train), (X_test, y_test) = imdb.load_data(num_
words=top_words)

np.load = np_load_old #revert back to default parameters
#it is a way around to deal with different versions of
libraries
```

Padding

Pad the sequences (reviews) that are shorter than the max_words.

PYTHON CODE:

```
X_train = sequence.pad_sequences(X_train, maxlen=max_words)
X_test = sequence.pad_sequences(X_test, maxlen=max_words)
```

14

Project 8: Activation Functions in NNs

Activation functions are an essential component of a Neural Network and help in learning nonlinear mappings that are otherwise impossible to learn with just linear functions. There are many activation functions that can be used in NN layers. Each one has its pros and cons. In this chapter, we are going to study a very popular activation function—RELU (Rectified Linear Unit), which clips the output of a neuron to zero in case its output is less than zero. It is mathematically defined as *max(0,x) *

14.1. Model Building

PYTHON CODE:

```
print ('Building The Model')

#Define the model type which is sequential in our case

model = Sequential()
#this is the syantax to add a layer and embedding is the layer
```

```
model.add(Embedding(top_words, EMBEDDING_DIM, input_
length=max_words))

#LSTM - Recurrent Layer)

model.add(LSTM(units = 32, dropout = 0.2, recurrent_dropout =
0.2))

#add a Dense layer with relu activation function
model.add(Dense(250, activation='relu'))

#add output layer
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])

print ('Model Built Successfully')
```

14.2. Model Summary

PYTHON CODE:

```
print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 500, 100)          500000
_________________________________________________________________
lstm_1 (LSTM)                (None, 32)                17024
_________________________________________________________________
dense_1 (Dense)              (None, 250)               8250
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 251
=================================================================
Total params: 525,525
Trainable params: 525,525
Non-trainable params: 0
_________________________________________________________________
None
```

14.3. Model Training

In this step, we will be training our model with the following configurations:

1. Epochs 20 — which means that our model will look at each example in our dataset 20 times.
2. Validation Split 0.3 — which means 30 percent of our training data will be used for validating the model.
3. Batch size 128 — which means there are going to be 128 examples used in each iteration from which our model will learn in a single step.

PYTHON CODE:

```
print ('Start Training')
model.fit(X_train, y_train, validation_split=0.3, epochs=20,
batch_size=128)
print ('Training Finished')
```

```
Start Training
Train on 17500 samples, validate on 7500 samples
Epoch 1/20
17500/17500 [==============================] - 111s 6ms/step - loss: 0.5225 - acc: 0.7366 - val_loss: 0.4036 - val_acc: 0.8173
Epoch 2/20
17500/17500 [==============================] - 110s 6ms/step - loss: 0.3794 - acc: 0.8374 - val_loss: 0.4830 - val_acc: 0.7801
Epoch 3/20
17500/17500 [==============================] - 110s 6ms/step - loss: 0.3234 - acc: 0.8665 - val_loss: 0.3859 - val_acc: 0.8357
Epoch 4/20
17500/17500 [==============================] - 110s 6ms/step - loss: 0.3052 - acc: 0.8750 - val_loss: 0.3904 - val_acc: 0.8324
Epoch 5/20
17500/17500 [==============================] - 110s 6ms/step - loss: 0.2811 - acc: 0.8885 - val_loss: 0.4196 - val_acc: 0.8088
Epoch 6/20
17500/17500 [==============================] - 110s 6ms/step - loss: 0.2740 - acc: 0.8882 - val_loss: 0.4271 - val_acc: 0.8177
Epoch 7/20
17500/17500 [==============================] - 110s 6ms/step - loss: 0.2402 - acc: 0.9049 - val_loss: 0.4119 - val_acc: 0.8380
Epoch 8/20
17500/17500 [==============================] - 110s 6ms/step - loss: 0.2288 - acc: 0.9101 - val_loss: 0.4184 - val_acc: 0.8284
Epoch 9/20
17500/17500 [==============================] - 110s 6ms/step - loss: 0.1947 - acc: 0.9249 - val_loss: 0.4874 - val_acc: 0.8245
Epoch 10/20
17500/17500 [==============================] - 110s 6ms/step - loss: 0.1747 - acc: 0.9321 - val_loss: 0.4757 - val_acc: 0.8141
Epoch 11/20
17500/17500 [==============================] - 110s 6ms/step - loss: 0.1505 - acc: 0.9429 - val_loss: 0.5348 - val_acc: 0.8036
Epoch 12/20
17500/17500 [==============================] - 110s 6ms/step - loss: 0.2145 - acc: 0.9117 - val_loss: 0.5135 - val_acc: 0.8091
Epoch 13/20
17500/17500 [==============================] - 110s 6ms/step - loss: 0.1662 - acc: 0.9346 - val_loss: 0.5398 - val_acc: 0.8217
Epoch 14/20
17500/17500 [==============================] - 109s 6ms/step - loss: 0.1411 - acc: 0.9449 - val_loss: 0.6341 - val_acc: 0.8143
Epoch 15/20
17500/17500 [==============================] - 110s 6ms/step - loss: 0.1212 - acc: 0.9546 - val_loss: 0.6286 - val_acc: 0.8032
Epoch 16/20
17500/17500 [==============================] - 110s 6ms/step - loss: 0.1095 - acc: 0.9593 - val_loss: 0.6380 - val_acc: 0.8128
Epoch 17/20
17500/17500 [==============================] - 110s 6ms/step - loss: 0.0990 - acc: 0.9626 - val_loss: 0.7176 - val_acc: 0.7961
Epoch 18/20
17500/17500 [==============================] - 110s 6ms/step - loss: 0.1024 - acc: 0.9610 - val_loss: 0.7101 - val_acc: 0.7985
Epoch 19/20
17500/17500 [==============================] - 110s 6ms/step - loss: 0.1021 - acc: 0.9628 - val_loss: 0.7319 - val_acc: 0.8111
Epoch 20/20
17500/17500 [==============================] - 110s 6ms/step - loss: 0.0824 - acc: 0.9707 - val_loss: 0.7655 - val_acc: 0.7991
Training Finished
```

14.4. Model Evaluation

PYTHON CODE:

```
print ('Evaluating the model on the Test Set')
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Evaluating the model on the Test Set
Accuracy: 80.22%
```

14.5. Using External Dataset

We already saw how to use the Keras preloaded dataset. Now, we are going to download and format an external dataset for our CNN.

PYTHON CODE:

```
#Download Large Movie Review Dataset and unzip it
!wget ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz
!tar xzf aclImdb_v1.tar.gz
```

```
--2019-07-06 05:30:51--  http://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz
Resolving ai.stanford.edu (ai.stanford.edu)... 171.64.68.10
Connecting to ai.stanford.edu (ai.stanford.edu)|171.64.68.10|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 84125825 (80M) [application/x-gzip]
Saving to: 'aclImdb_v1.tar.gz'

aclImdb_v1.tar.gz   100%[===================>]  80.23M  21.7MB/s    in 3.7s

2019-07-06 05:30:55 (21.7 MB/s) - 'aclImdb_v1.tar.gz' saved [84125825/84125825]
```

14.6. Prepare the Dataset

In this step, we will be reading the files and appending them to a single Pandas Dataframe so that we can use it for training and testing our model.

PYTHON CODE:

```
#lets format the dataset to our purpose
#where dataset is extracted
folder = 'aclImdb'
#two classes
labels = {'pos': 1, 'neg': 0}
#two splits in dataset
splits = ('test', 'train')
#create a pandas dataframe
data_frame = pd.DataFrame()

for split in splits:
    for label in ('pos', 'neg'):
        #join using path.join as it is independent of os
        path = os.path.join(folder, split, label)
        #list all the directories in the path
        for file in os.listdir (path) :
            with open(os.path.join(path, file),'r',
encoding='utf-8') as infile:
                txt = infile.read()
            data_frame = data_frame.append([[txt,
labels[label]]],ignore_index=True)

data_frame.columns = ['review', 'sentiment']
```

14.7. Analyze the Dataset

We will be looking at a few records from our dataset, the total number of records in our dataset, and the class distribution (number of positive and negative sentiments) in our dataset.

PYTHON CODE:

```
#lets analyze the transformed dataset
data_frame.head()
```

	review	sentiment
0	From a perspective that it is possible to make...	1
1	I would say to the foreign people who have see...	1
2	Less Than Zero could have been the 80s movie t...	1
3	I just watched this movie on it's premier nigh...	1
4	I had been interested in this film for a long ...	1

PYTHON CODE:

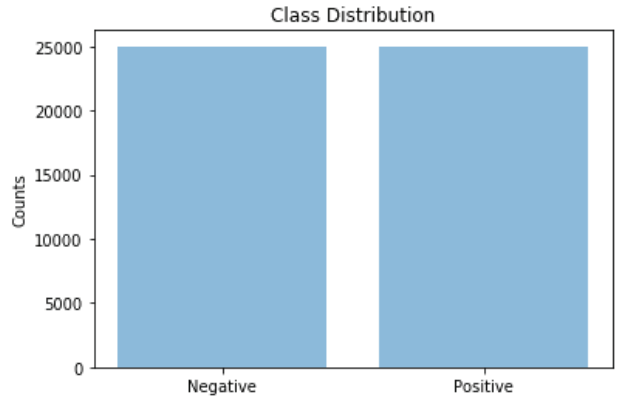
```
print ('Total Records = ' , len(data_frame))
```

```
Total Records =  50000
```

PYTHON CODE:

```
#to plot the bar chart
import matplotlib.pyplot as plt

print ('Class Distribution')

sentiments = ('Negative', 'Positive')
y_pos = np.arange(len(sentiments))
counts = [data_frame.sentiment.value_counts()[0], data_frame.
sentiment.value_counts()[1]]


plt.bar(y_pos, counts, align='center', alpha=0.5)
plt.xticks(y_pos, sentiments)
plt.ylabel('Counts')
plt.title('Class Distribution')

plt.show()
```

Class Distribution



PYTHON CODE:

```
#Lets prepare the dataset for Conv Network without using imdb
API from Keras
x_train_manual = data_frame.loc[:24999, 'review'].values
y_train_manual = data_frame.loc[:24999, 'sentiment'].values
x_test_manual = data_frame.loc[25000:, 'review'].values
y_test_manual = data_frame.loc[25000:, 'sentiment'].values

#Reconfirm class distribution in train and test sets after
split
print ('Distribution in train set ', np.unique(y_train_manual,
return_counts= True))
print ('Distribution in test set ', np.unique(y_test_manual,
return_counts= True))
```

```
[ ]  #Lets prepare the dataset for Conv Network without using imdb API from Keras
     x_train_manual = data_frame.loc[:24999, 'review'].values
     y_train_manual = data_frame.loc[:24999, 'sentiment'].values
     x_test_manual = data_frame.loc[25000:, 'review'].values
     y_test_manual = data_frame.loc[25000:, 'sentiment'].values
```

```
[ ]  #Reconfirm class distribution in train and test sets after split
     print ('Distribution in train set ', np.unique(y_train_manual, return_counts= True))
     print ('Distribution in test set ', np.unique(y_test_manual, return_counts= True))
```

```
Distribution in train set  (array([0, 1]), array([12500, 12500]))
Distribution in test set  (array([0, 1]), array([12500, 12500]))
```

14.8. Dataset Formatting

In this step, we are going to convert our text sequence data into number sequences.

1. Initialize Tokenizer Object with top_words (as explained above to use only the most frequent words (top_words) and make the rest zero), and fit on our training data.
2. Convert the text sequences to number sequences by function texts_to_sequences, and pass the text data.
3. Pad the sequences that are less than the maxlen (max_words) by padding them.

PYTHON CODE:

```
tok_train = Tokenizer(num_words=top_words)
tok_train.fit_on_texts(x_train_manual)

x_train_manual = tok_train.texts_to_sequences(x_train_manual)
x_test_manual = tok_train.texts_to_sequences(x_test_manual)

x_train_pad = sequence.pad_sequences(x_train_
manual,maxlen=max_words)
x_test_pad = sequence.pad_sequences(x_test_manual,maxlen=max_
words)
```

14.9. Model Building

PYTHON CODE:

```
#import conv and max_pool layer for the CNN model
from keras.layers import Conv1D, MaxPool1D

#Now We will build the Convolutional Model

print ('Building the CNN model')
model_conv = Sequential()
model_conv.add(Embedding(top_words, EMBEDDING_DIM, input_
length=max_words))
```

```
model_conv.add(Conv1D(filters=32, kernel_size=8,
activation='relu'))

model_conv.add(MaxPool1D(pool_size=2)) #pooling layer to
reduce dimension
model_conv.add(Flatten())
 #flatten the output from pool before passing to dense layer

model_conv.add(Dense(36, activation='relu'))
model_conv.add(Dense(1, activation='sigmoid'))

model_conv.compile(loss='binary_crossentropy',
optimizer='adam', metrics=['accuracy'])

print ('CNN model build successfully')
```

```
Building the CNN model
CNN model built successfully.
```

14.10. Model Summary

PYTHON CODE:

```
print(model_conv.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 500, 100)          500000
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 493, 32)           25632
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 246, 32)           0
_________________________________________________________________
flatten_1 (Flatten)          (None, 7872)              0
_________________________________________________________________
dense_3 (Dense)              (None, 36)                283428
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 37
=================================================================
Total params: 809,097
Trainable params: 809,097
Non-trainable params: 0
_________________________________________________________________
None
```

14.11. Model Training

PYTHON CODE:

```
print ('Start Model Training')
model_conv.fit(x_train_pad, y_train_manual, validation_
split=0.3, epochs=10, batch_size=128)
print ('Model Training finished')
```

```
Start Model Training
Train on 17500 samples, validate on 7500 samples
Epoch 1/10
17500/17500 [==============================] - 58s 3ms/step - loss: 0.4743 - acc: 0.7766 - val_loss: 0.8672 - val_acc: 0.7063
Epoch 2/10
17500/17500 [==============================] - 58s 3ms/step - loss: 0.3034 - acc: 0.9084 - val_loss: 1.0008 - val_acc: 0.6851
Epoch 3/10
17500/17500 [==============================] - 58s 3ms/step - loss: 0.2419 - acc: 0.9379 - val_loss: 0.9103 - val_acc: 0.7919
Epoch 4/10
17500/17500 [==============================] - 58s 3ms/step - loss: 0.1954 - acc: 0.9594 - val_loss: 1.0943 - val_acc: 0.7759
Epoch 5/10
17500/17500 [==============================] - 58s 3ms/step - loss: 0.1629 - acc: 0.9746 - val_loss: 1.1150 - val_acc: 0.8005
Epoch 6/10
17500/17500 [==============================] - 58s 3ms/step - loss: 0.1410 - acc: 0.9819 - val_loss: 1.3341 - val_acc: 0.7672
Epoch 7/10
17500/17500 [==============================] - 58s 3ms/step - loss: 0.1253 - acc: 0.9857 - val_loss: 1.1961 - val_acc: 0.8051
Epoch 8/10
17500/17500 [==============================] - 59s 3ms/step - loss: 0.1129 - acc: 0.9877 - val_loss: 1.5034 - val_acc: 0.7697
Epoch 9/10
17500/17500 [==============================] - 58s 3ms/step - loss: 0.1032 - acc: 0.9887 - val_loss: 1.5090 - val_acc: 0.7749
Epoch 10/10
17500/17500 [==============================] - 58s 3ms/step - loss: 0.0953 - acc: 0.9894 - val_loss: 1.6581 - val_acc: 0.7616
Model Training finished
```

14.12. Model Evaluation

PYTHON CODE:

```
# Final evaluation of the model

print ('Evaluating the model')
scores = model_conv.evaluate(x_test_pad, y_test_manual,
verbose=0)

print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Evaluating the model
Accuracy: 84.51%
```

14.13. Testing Samples on the CNN Model

Let's test our model on the same reviews.

The resulting probabilities of positive reviews tend to be closer to 1, as it is the positive class.

PYTHON CODE:

```
sample1 = "awesome, must watch movie will definitely watch
again if I get the chance"
sample2 = "movie was really good, really liked the acting as
well"
sample3 = "movie was not that good at most OK but I did like
the acting"
sample4 = "movie was so horrible, very bad acting as well how
can people watch this crap"
sample5 = "i have not seen a movie worse than this one"

#make a list of lists
test_sample = [sample1, sample2, sample3, sample4, sample5]
#make sequences just as in done earlier steps
test_sample_tok =  tok_train.texts_to_sequences(test_sample)
#pad the sequences as done in earlier steps
test_sample_pad =  sequence.pad_sequences (test_sample_tok,
maxlen=max_words)

print (model_conv.predict(test_sample_pad))
```

```
[[0.9980488 ]
 [0.9915957 ]
 [0.28260344]
 [0.23432045]
 [0.23432045]]
```

14.14. Testing Samples on the LSTM Model

This is a bit different from the above code because we have to use the Keras tokenizer as we did not build the tokenizer for the preloaded dataset.

PYTHON CODE:

```
sample1 = “awesome, must watch movie will difinetly watch
again”
sample2 = “movie was awesome”
sample3 = “movie was not that good at most OK but I did like
the acting”
sample4 = “movie was so horrible, very bad acting as well how
can people watch this crap”
sample5 = “bad indeed”

test_sample = [sample1, sample2, sample3, sample4, sample5]

#tk = Tokenizer(nb_words=top_words)
#tk.fit_on_texts(test_sample)

sequences = tok_train.texts_to_sequences(test_sample)
padded_sequences = sequence.pad_sequences(sequences,
maxlen=max_words)

prediction = model.predict(padded_sequences)
print(prediction)
```

```
[ ]  sample1 = "awesome, must watch movie will difinetly watch again"
     sample2 = "movie was awesome"
     sample3 = "movie was not that good at most OK but I did like the acting"
     sample4 = "movie was so horrible, very bad acting as well how can people watch this crap"
     sample5 = "bad indeed"

     test_sample = [sample1, sample2, sample3, sample4, sample5]

     #tk = Tokenizer(nb_words=top_words)
     #tk.fit_on_texts(test_sample)

     sequences = tok_train.texts_to_sequences(test_sample)
     padded_sequences = sequence.pad_sequences(sequences, maxlen=max_words)

     prediction = model.predict(padded_sequences)
     print(prediction)
```

```
[[0.9996575 ]
 [0.99932814]
 [0.7962845 ]
 [0.11646644]
 [0.0058602 ]]
```

Congrats on Completing This Book

Hopefully, this book has demystified the notion of deep learning. But this is just the beginning. Now that you are familiar with the logic behind the different deep learning algorithms, understand the role of linear algebra, probability, and statistics, and know how to create simple algorithms, it is time to move on. We have more books and courses to reinforce your efforts and guide you at every stage.

Deep learning is a crucial development in today's world. The concepts behind it have been around for more than a decade, but the age of deep learning, AI, and related models—such as artificial intelligence, data science, and more—is now. The change is just happening, and it is fast.

You have made a great decision to start your journey into the world of data science and deep learning with this book. Today, the knowledge and the ability to use data science and AI is a competitive advantage. Tomorrow, it will be a mere necessity.

Deep learning techniques have already started to change the world of business by creating new value for data. The future will be even more exciting. Very soon, most of the devices and apps that we use daily will be fueled by deep learning

algorithms. Many of them already are. Now you have a chance to become a part of this major development. Congrats on your decision, and don't forget to check out our other books!

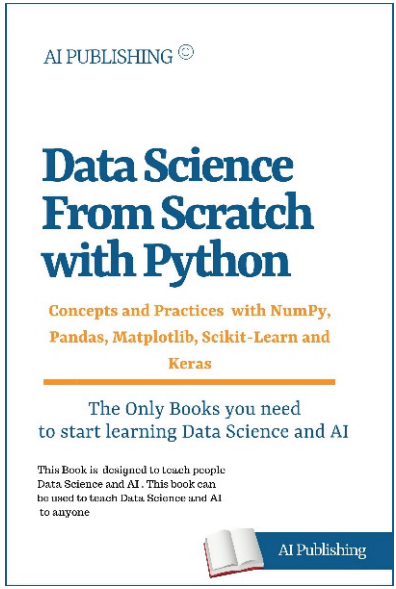
If you want to help us produce more material like this, then please leave an honest review. It really does make a difference.

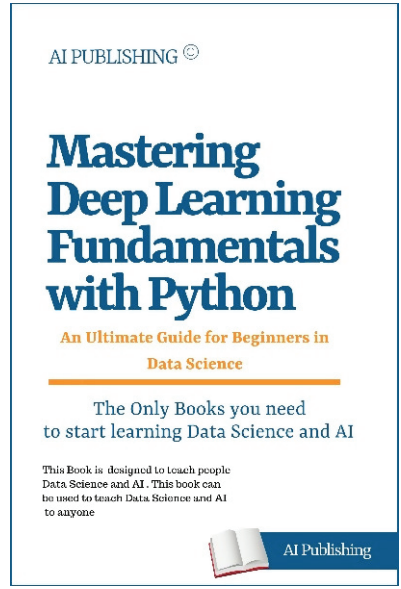
If you have any feedback, please let us know by sending an email to contact@aispublishing.net.

Your feedback is highly valued. We look forward to hearing from you as it will be very helpful for us to improve the quality of our books.

Until next time, happy analyzing.

From the same Publisher





AI PUBLISHING