|  |
| --- |
| HMR Institute of technology and management |
| X Ray Image Processing |
| Project Report |
|  |
| **Ujjwal Mittal** |
| **7/1/2018** |

|  |
| --- |
| Analyzed and Processed XRay Images through Deep learning: Convolution Neural Network (Keras) |

Table of Content

Abstract2

Algorithms Used3

**Libraries Used4**

Functions Used5

Dataset6

**Codes Description7**

Variable Explorer8

IPython Console9

**Code7**

Summary

Team Details

**Chapter – 2**

Abstract

Technology is seeping deep in every industry, and with the advent of new technologies like Machine Learning and Deep Learning, a question that arises is how every industry can benefit and excel in its field using the said technologies. Medicine and health is a field where it is pivotal to keep updating and improving the existing technologies and methodologies for better treatment purposes. We built a system to incorporate the various concepts of Machine and Deep Learning in health diagnosis.

We built an Image Processing System for Digital X-Ray Images to study the lungs of a person and predict if it is normal or infected with pneumonia. Also, if infected, the system helps diagnose whether the infection is bacterial pneumonia or viral pneumonia.

Machine learning is a subset of artificial intelligence in the field of computer science that often uses statistical techniques to give computers the ability to "learn" (i.e., progressively improve performance on a specific task) with data, without being explicitly programmed. Or in other words, we train the algorithm with a set of data, and on the basis of the patterns formed on the dataset, the algorithm helps in predicting a requisite data.

We have used Machine Learning and Deep Learning to build the algorithm for this project. The dataset has various X-Ray images of the lungs. The normal chest X-ray depicts clear lungs without any areas of abnormal opacification in the image. Bacterial pneumonia typically exhibits a focal lobar consolidation, whereas viral pneumonia manifests with a more diffuse ‘‘interstitial’’ pattern in both lungs. These are the patterns that the ML algorithm builds on to make predictions on the test set.

**Chapter – 3**

Algorithms Used

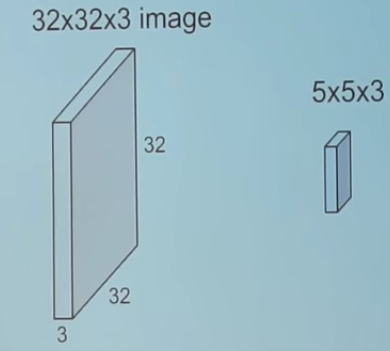
Convolution Neural Network

Convolutional Neural Networks are very similar to ordinary Neural Networks – they are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole network expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other. And they have a loss function (e.g. SVM/Softmax) on the last (fully-connected) layer.

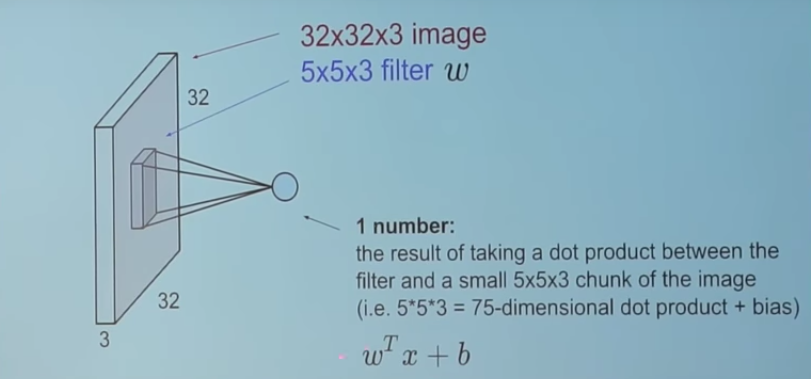
The main difference is that CNN architectures make the explicit assumption that the inputs are images, which allows us to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the amount of parameters in the network. In simpler words, unlike regulat neural networks, where the input is a vector, here the input is a multi-channeled image (3 channeled in this case).

CNN has 4 main steps – Convolution, Max Pooling, Flattening, and Full Connection.

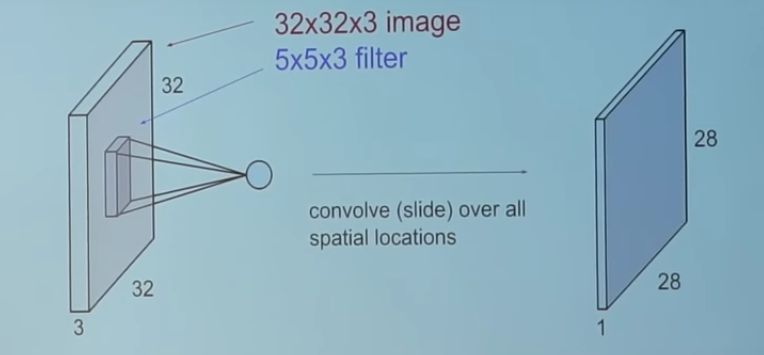
First, let us understand what exactly Convolution means.



We take the 5\*5\*3 filter and slide it over the complete image and along the way take the dot product between the filter and chunks of the input image.



For every dot product taken, the result is a scalar. So, what happens when we convolve the complete image with the filter? This!

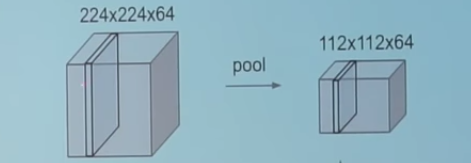


The convolution layer is the main building block of a convolutional neural network. The convolution layer comprises of a set of independent filters. Each filter is independently convolved with the image and we end up with feature maps. For a particular feature map *(the output received on convolving the image with a particular filter is called a feature map)*, each neuron is connected only to a small chunk of the input image and all the neurons have the same connection weights.

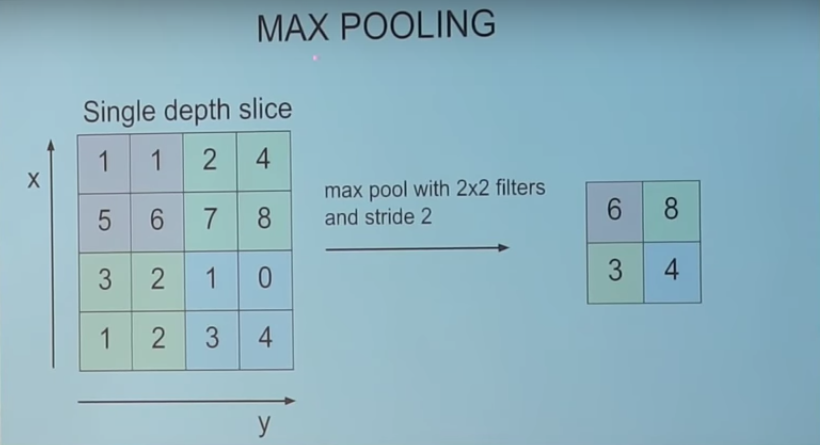
CNNs have a couple of concepts called parameter sharing and local connectivity. Parameter sharing is sharing of weights by all neurons in a particular feature map whereas local connectivity is the concept of each neural connected only to a subset of the input image (unlike a neural network where all the neurons are fully connected). This helps to reduce the number of parameters in the whole system and makes the computation more efficient.

Now, let us discuss about Pooling Layers.

It is another building block of a CNN.



Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network. Pooling layer operates on each feature map independently. The most common approach used in pooling is *max pooling.*



The FC is the fully connected layer of neurons at the end of CNN. Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular Neural Networks and work in a similar way.

Convolutional networks were inspired by biological processes in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field.

CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns the filters that in traditional algorithms were hand-engineered. This independence from prior knowledge and human effort in feature design is a major advantage.

They have applications in image and video recognition, recommender systems and natural language processing. Let us discuss these applications in detail –

* Image recognition

CNNs are often used in image recognition systems. In 2012 an error rate of 0.23 percent on the MNIST database was reported. Another paper on using CNN for image classification reported that the learning process was "surprisingly fast"; in the same paper, the best published results as of 2011 were achieved in the MNIST database and the NORB database.

When applied to facial recognition, CNNs achieved a large decrease in error rate. Another paper reported a 97.6 percent recognition rate on "5,600 still images of more than 10 subjects". CNNs were used to assess video quality in an objective way after manual training; the resulting system had a very low root mean square error.

* Video Analysis

Compared to image data domains, there is relatively little work on applying CNNs to video classification. Video is more complex than images since it has another (temporal) dimension. However, some extensions of CNNs into the video domain have been explored. One approach is to treat space and time as equivalent dimensions of the input and perform convolutions in both time and space. Another way is to fuse the features of two convolutional neural networks, one for the spatial and one for the temporal stream. Unsupervised learning schemes for training spatio-temporal features have been introduced, based on Convolutional Gated Restricted Boltzmann Machines and Independent Subspace Analysis.

* Natural language processing

CNNs have also explored natural language processing. CNN models are effective for various NLP problems and achieved excellent results in semantic parsing, search query retrieval, sentence modeling, classification, prediction and other traditional NLP tasks

* Drug discovery

CNNs have been used in drug discovery. Predicting the interaction between molecules and biological proteins can identify potential treatments. In 2015, Atomwise introduced AtomNet, the first deep learning neural network for structure-based rational drug design. The system trains directly on 3-dimensional representations of chemical interactions. Similar to how image recognition networks learn to compose smaller, spatially proximate features into larger, complex structures, AtomNet discovers chemical features, such as aromaticity, sp3 carbons and hydrogen bonding. Subsequently, AtomNet was used to predict novel candidate biomolecules for multiple disease targets, most notably treatments for the Ebola virus and multiple sclerosis.

**Chapter – 4**

Libraries Used

4.1 Numpy

NumPy is a library for the [Python programming language](https://en.wikipedia.org/wiki/Python_(programming_language)), adding support for large, multi-dimensional [arrays](https://en.wikipedia.org/wiki/Array_data_structure) and [matrices](https://en.wikipedia.org/wiki/Matrix_(math)), along with a large collection of [high-level](https://en.wikipedia.org/wiki/High-level_programming_language)[mathematical](https://en.wikipedia.org/wiki/Mathematics) [functions](https://en.wikipedia.org/wiki/Function_(mathematics)) to operate on these arrays. The ancestor of NumPy, Numeric, was originally created by [Jim Hugunin](https://en.wikipedia.org/wiki/Jim_Hugunin)with contributions from several other developers. In 2005, [Travis Oliphant](https://en.wikipedia.org/wiki/Travis_Oliphant) created NumPy by incorporating features of the competing Numarray into Numeric, with extensive modifications. NumPy is [open-source software](https://en.wikipedia.org/wiki/Open-source_software) and has many contributors.

4.1.1 History

The Python programming language was not initially designed for numerical computing, but attracted the attention of the scientific and engineering community early on, so that a special interest group called matrix-sig was founded in 1995 with the aim of defining an array computing package. Among its members was Python designer and maintainer Guido van Rossum, who implemented extensions to Python's syntax (in particular the indexing syntax) to make array computing easier.

An implementation of a matrix package was completed by Jim Fulton, then generalized by Jim Hugunin to become Numeric, also variously called Numerical Python extensions or NumPy. Hugunin, a graduate student at MIT, joined CNRI to work on JPython in 1997 leaving Paul Dubois of LLNL to take over as maintainer. Other early contributors include David Ascher, Konrad Hinsen and Travis Oliphant.

4.1.2 Traits

NumPy targets the CPython reference implementation of Python, which is a non-optimizing bytecode interpreter. Mathematical algorithms written for this version of Python often run much slower than compiled equivalents. NumPy addresses the slowness problem partly by providing multidimensional arrays and functions and operators that operate efficiently on arrays, requiring rewriting some code, mostly inner loops using NumPy.

4.1.3 Limitations

Inserting or appending entries to an array is not as trivially possible as it is with Python's lists. The np.pad(...) routine to extend arrays actually creates new arrays of the desired shape and padding values, copies the given array into the new one and returns it. NumPy's np.concatenate([a1,a2]) operation does not actually link the two arrays but returns a new one, filled with the entries from both given arrays in sequence. Reshaping the dimensionality of an array with np.reshape(...) is only possible as long as the number of elements in the array does not change. These circumstances originate from the fact that NumPy's arrays must be views on contiguous memory buffers. A replacement package called Blaze attempts to overcome this limitation.

4.2 Matplotlib

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK+. There is also a procedural "pylab" interface based on a state machine (like OpenGL), designed to closely resemble that of MATLAB, though its use is discouraged. SciPy makes use of matplotlib.

Matplotlib was originally written by John D. Hunter, as an active development community, and is distributed under a BSD-style license. Michael Droettboom was nominated as matplotlib's lead developer shortly before John Hunter's death in 2012, and further joined by Thomas Caswell.

Pyplot is a matplotlib module which provides a MATLAB-like interface. Matplotlib is designed to be as usable as MATLAB, with the ability to use Python, and the advantage of being free and open-source.

4.3 Pandas

In computer programming, pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license. The name is derived from the term "panel data", an econometrics term for data sets that include observations over multiple time periods for the same individuals.

4.3.1 History

Developer Wes McKinney started working on pandas in 2008 while at AQR Capital Management out of the need for a high performance, flexible tool to perform quantitative analysis on financial data. Before leaving AQR he was able to convince management to allow him to open source the library.

Another AQR employee, Chang She, joined the effort in 2012 as the second major contributor to the library. In 2015, pandas signed on as a fiscally sponsored project of NumFOCUS, a 501(c)(3) nonprofit charity in the United States

4.3.2 Library Features

* DataFrame object for data manipulation with integrated indexing.
* Tools for reading and writing data between in-memory data structures and different file formats.
* Data alignment and integrated handling of missing data.
* Reshaping and pivoting of data sets.
* Label-based slicing, fancy indexing, and subsetting of large data sets.
* Data structure column insertion and deletion.
* Group by engine allowing split-apply-combine operations on data sets.
* Data set merging and joining.
* Hierarchical axis indexing to work with high-dimensional data in a lower-dimensional data structure.
* Time series-functionality: Date range generation[3] and frequency conversion, moving window statistics, moving window linear regressions, date shifting and lagging.
* The library is highly optimized for performance, with critical code paths written in Cython or C.

4.4 Keras

Keras is an open source neural network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit or Theano. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. It was developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System), and its primary author and maintainer is François Chollet, a Google engineer.

In 2017, Google's TensorFlow team decided to support Keras in TensorFlow's core library. Chollet explained that Keras was conceived to be an interface rather than a standalone machine-learning framework. It offers a higher-level, more intuitive set of abstractions that make it easy to develop deep learning models regardless of the computational backend used. Microsoft added a CNTK backend to Keras as well, available as of CNTK v2.0.

4.4.1 Features

Keras contains numerous implementations of commonly used neural network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier. The code is hosted on GitHub, and community support forums include the GitHub issues page, and a Slack channel.

Keras allows users to productize deep models on smartphones (iOS and Android), on the web, or on the Java Virtual Machine. It also allows use of distributed training of deep learning models on clusters of Graphics Processing Units (GPU).

4.5 TensorFlow

TensorFlow is an open-source software library for dataflow programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks.[3] It is used for both research and production at Google, often replacing its closed-source predecessor, DistBelief.

TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache 2.0 open source license on November 9, 2015. Tensorflow is just a computational framework for expressing algorithms involving large number of Tensor operations, since Neural networks can be expressed as computational graphs they can be implemented using Tensorflow as a series of operations on Tensors. Tensors are N-dimensional matrices which represents our Data.

4.5.1 History

Starting in 2011, Google Brain built DistBelief as a proprietary machine learning system based on deep learning neural networks. Its use grew rapidly across diverse Alphabet companies in both research and commercial applications. Google assigned multiple computer scientists, including Jeff Dean, to simplify and refactor the codebase of DistBelief into a faster, more robust application-grade library, which became TensorFlow. In 2009, the team, led by Geoffrey Hinton, had implemented generalized backpropagation and other improvements which allowed generation of neural networks with substantially higher accuracy, for instance a 25% reduction in errors in speech recognition.

TensorFlow is Google Brain's second-generation system. Version 1.0.0 was released on February 11, 2017. While the reference implementation runs on single devices, TensorFlow can run on multiple CPUs and GPUs (with optional CUDA and SYCL extensions for general-purpose computing on graphics processing units). TensorFlow is available on 64-bit Linux, macOS, Windows, and mobile computing platforms including Android and iOS.

4.5.2 Applications

TensorFlow provides stable Python API and C APIs and without API backwards compatibility guarantee: C++, Go, Java, JavaScript and Swift (early release). Third party packages are available for C#, Haskell, Julia, R, Scala, Rust, and OCaml. Among the applications for which TensorFlow is the foundation, are automated image captioning software, such as DeepDream. RankBrain now handles a substantial number of search queries, replacing and supplementing traditional static algorithm-based search results.

**Chapter – 5**

Function Used

5.1.Path

Used to set the path of directory in which images/data are stored

class Path(PurePath):

Initialize self. See help(type(self)) for accurate signature.

5.2.Glob

Definition**:** glob(Pattern)

Type**:** Present in Pathlib.Path Module

Iterate over this subtree and yield all existing files (of any kind, including directories) matching the given pattern.

5.3.Append

Definition**:** append(other, ignore\_index=False, verify\_integrity=False, sort=None)

Type**:** Function of pandas.core.frame module

Append rows of other to the end of this frame, returning a new object. Columns not in this frame are added as new columns.

5.3.1.Parameter

other **:**DataFrame or Series/dict-like object, or list of these

The data to append.

ignore\_index**:**boolean, default False

If True, do not use the index labels.

verify\_integrity**:**boolean, default False

If True, raise ValueError on creating index with duplicates.

sort **:**boolean, default None

Sort columns if the columns of self and other are not aligned. The default sorting is deprecated and will change to not-sorting in a future version of pandas. Explicitly pass sort=True to silence the warning and sort. Explicitly pass sort=False to silence the warning and not sort.

5.3.2.Returns

appended : DataFrame

5.3.3.Notes

If a list of dict/series is passed and the keys are all contained in the DataFrame’s index, the order of the columns in the resulting DataFrame will be unchanged.

Iteratively appending rows to a DataFrame can be more computationally intensive than a single concatenate. A better solution is to append those rows to a list and then concatenate the list with the original DataFrame all at once.

5.3.4.See also

pandas.concat**:**General function to concatenate DataFrame, Seriesor Panel objects.

5.4.DataFrame

Defination**:** DataFrame(rows and colunms)

Type: Function of pandas.core.frame module

Two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). Arithmetic operations align on both row and column labels. Can be thought of as a dict-like container for Series objects. The primary pandas data structure.

5.4.1.Parameters

data **:**numpy ndarray (structured or homogeneous), dict, or DataFrame

Dict can contain Series, arrays, constants, or list-like objects

index : Index or array-like

Index to use for resulting frame. Will default to RangeIndex if no indexing information part of input data and no index provided

columns : Index or array-like

Column labels to use for resulting frame. Will default to RangeIndex (0, 1, 2, …, n) if no column labels are provided

dtype : dtype, default None

Data type to force. Only a single dtype is allowed. If None, infer

copy : boolean, default False

Copy data from inputs. Only affects DataFrame / 2d ndarray input

5.4.2.See also

DataFrame.from\_records : constructor from tuples, also record arrays DataFrame.from\_dict : from dicts of Series, arrays, or dicts DataFrame.from\_items : from sequence of (key, value) pairs pandas.read\_csv, pandas.read\_table, pandas.read\_clipboard

5.5.Sample

Definition**:** sample(n=None, frac=None, replace=False, weight=None, random\_state=None, axis=None)

Type: Function of pandas.core.generic module

Return a random sample of items from an axis of object.

You can use random\_state for reproducibility.

5.5.1.Parameters

, (optional)

Number of items from axis to return. Cannot be used with frac. Default = 1 if frac = None.

frac : float, optional

Fraction of axis items to return. Cannot be used with n.

replace : boolean, optional

Sample with or without replacement. Default = False.

weights : str or ndarray-like, optional

Default ‘None’ results in equal probability weighting. If passed a Series, will align with target object on index. Index values in weights not found in sampled object will be ignored and index values in sampled object not in weights will be assigned weights of zero. If called on a DataFrame, will accept the name of a column when axis = 0. Unless weights are a Series, weights must be same length as axis being sampled. If weights do not sum to 1, they will be normalized to sum to 1. Missing values in the weights column will be treated as zero. inf and -inf values not allowed.

random\_state : int or numpy.random.RandomState, optional

n : int

Seed for the random number generator (if int), or numpy RandomState object.

axis : int or string, optional

Axis to sample. Accepts axis number or name. Default is stat axis for given data type (0 for Series and DataFrames, 1 for Panels).

5.5.2.Returns

A new object of same type as caller.

5.6.Reset\_index

Definition**:** reset\_index(self, level=None, drop=False, inplace=False, col\_level=0, col\_fill="")

Type: Method of Frame class in pandas.core.frame module

For DataFrame with multi-level index, return new DataFrame with labeling information in the columns under the index names, defaulting to ‘level\_0’, ‘level\_1’, etc. if any are None. For a standard index, the index name will be used (if set), otherwise a default ‘index’ or ‘level\_0’ (if ‘index’ is already taken) will be used.

5.6.1.Parameters

level : int, str, tuple, or list, default None

Only remove the given levels from the index. Removes all levels by default

drop : boolean, default False

Do not try to insert index into dataframe columns. This resets the index to the default integer index.

inplace : boolean, default False

Modify the DataFrame in place (do not create a new object)

col\_level : int or str, default 0

If the columns have multiple levels, determines which level the labels are inserted into. By default it is inserted into the first level.

col\_fill : object, default ‘’

If the columns have multiple levels, determines how the other levels are named. If None then the index name is repeated.

5.6.1.Returns

resetted : DataFrame

5.7.len

Definition**:** len(...)

Type: Function of buildins module

Return the number of items in a container.

5.7.Sequential

Definition**:** Sequential(layers=None, name=None)

Type**:** Present in keras.engine.sequential module

class Sequential(Model):

Linear stack of layers.

5.7.1.Arguments

layers: list of layers to add to the model.

# Example

[``](#id1)python # Optionally, the first layer can receive an `input\_shape argument: model = Sequential() model.add(Dense(32, input\_shape=(500,)))

# Afterwards, we do automatic shape inference: model.add(Dense(32))

# This is identical to the following: model = Sequential() model.add(Dense(32, input\_dim=500))

# And to the following: model = Sequential() model.add(Dense(32, batch\_input\_shape=(None, 500)))

# Note that you can also omit the input\_shape argument: # In that case the model gets built the first time you call fit (or other # training and evaluation methods). model = Sequential() model.add(Dense(32)) model.add(Dense(32)) model.compile(optimizer=optimizer, loss=loss)

# This builds the model for the first time: model.fit(x, y, batch\_size=32, epochs=10)

# Note that when using this delayed-build pattern # (no input shape specified), # the model doesn’t have any weights until the first call # to a training/evaluation method (since it isn’t yet built): model = Sequential() model.add(Dense(32)) model.add(Dense(32)) model.weights # returns []

# Whereas if you specify the input shape, the model gets built continuously # as you are adding layers: model = Sequential() model.add(Dense(32, input\_shape=(500,))) model.add(Dense(32)) model.weights # returns list of length 4

# When using the delayed-build pattern (no input shape specified), you can # choose to manually build your model by calling # build(batch\_input\_shape): model = Sequential() model.add(Dense(32)) model.add(Dense(32)) model.build((None, 500)) model.weights # returns list of length 4 [``](#id3)[`](#id5)

5.8.ADD

Definition**:** add(self, layer)

Type**:** Method of Sequential class in keras.engine.sequential module

Adds a layer instance on top of the layer stack.

5.8.1.Arguments

layer: layer instance.

5.8.2.Raises

TypeError: If *layer* is not a layer instance. ValueError: In case the*layer* argument does not

know its input shape.

ValueError: In case the layer argument has

multiple output tensors, or is already connected somewhere else (forbidden in Sequential models).

5.9.Conv2D

Type**:** Class in keras.layes.convolution module

padding=’valid’,

data\_format=None, dilation\_rate=(1, 1), activation=None, use\_bias=True, kernel\_initializer=’glorot\_uniform’, bias\_initializer=’zeros’, kernel\_regularizer=None, bias\_regularizer=None, activity\_regularizer=None, kernel\_constraint=None, bias\_constraint=None, [\*\*](#id1)kwargs)

2D convolution layer (e.g. spatial convolution over images).

This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs. If use\_bias is True, a bias vector is created and added to the outputs. Finally, if activation is not None, it is applied to the outputs as well.

When using this layer as the first layer in a model, provide the keyword argument input\_shape (tuple of integers, does not include the sample axis), e.g. input\_shape=(128, 128, 3) for 128x128 RGB pictures indata\_format=”channels\_last”.

5.9.1.Arguments

filters: Integer, the dimensionality of the output space

(i.e. the number of output filters in the convolution).

kernel\_size: An integer or tuple/list of 2 integers, specifying the

height and width of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.

strides: An integer or tuple/list of 2 integers,

specifying the strides of the convolution along the height and width. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value != 1 is incompatible with specifying any *dilation\_rate*value != 1.

padding: one of “valid” or “same” (case-insensitive).

data\_format: A string,

one of *“channels\_last”* or *“channels\_first”*. The ordering of the dimensions in the inputs. *“channels\_last”* corresponds to inputs with shape *(batch, height, width, channels)* while*“channels\_first”* corresponds to inputs with shape *(batch, channels, height, width)*. It defaults to the*image\_data\_format* value found in your Keras config file at *~/.keras/keras.json*. If you never set it, then it will be “channels\_last”.

dilation\_rate: an integer or tuple/list of 2 integers, specifyingthe dilation rate to use for dilated convolution. Can be a single integer to specify the same value for all spatial dimensions. Currently, specifying any dilation\_rate value != 1 is incompatible with specifying any stride value != 1.

activation: Activation function to use

(see [activations](../activations.md)). If you don’t specify anything, no activation is applied (ie. “linear” activation:*a(x) = x*).

use\_bias: Boolean, whether the layer uses a bias vector. kernel\_initializer: Initializer for the *kernel* weights matrix

bias\_initializer: Initializer for the bias vector

(see [initializers](../initializers.md)).

kernel\_regularizer: Regularizer function applied to

the kernel weights matrix (see [regularizer](../regularizers.md)).

bias\_regularizer: Regularizer function applied to the bias vector

(see [regularizer](../regularizers.md)).

activity\_regularizer: Regularizer function applied to

the output of the layer (its “activation”). (see [regularizer](../regularizers.md)).

kernel\_constraint: Constraint function applied to the kernel matrix

(see [constraints](../constraints.md)).

bias\_constraint: Constraint function applied to the bias vector

(see [constraints](../constraints.md)).

5.9.2.Input shape

4D tensor with shape: (samples, channels, rows, cols) ifdata\_format is “channels\_first” or 4D tensor with shape: (samples, rows, cols, channels) if data\_format is “channels\_last”.

5.9.3.Output shape

4D tensor with shape: (samples, filters, new\_rows, new\_cols) if data\_format is “channels\_first” or 4D tensor with shape: (samples, new\_rows, new\_cols, filters) if data\_format is “channels\_last”. rowsand cols values might have changed due to padding.

5.10.MaxPooling2D

Type: Class in keras.layers.pooling module

strides=None, padding=’valid’, data\_format=None, [\*\*](#id1)kwargs)

Max pooling operation for spatial data.

5.10.1.Arguments

pool\_size: integer or tuple of 2 integers,

factors by which to downscale (vertical, horizontal). (2, 2) will halve the input in both spatial dimension. If only one integer is specified, the same window length will be used for both dimensions.

strides: Integer, tuple of 2 integers, or None.

Strides values. If None, it will default to pool\_size.

padding: One of “valid” or “same” (case-insensitive). data\_format: A string,

one of channels\_last (default) or channels\_first. The ordering of the dimensions in the inputs. channels\_lastcorresponds to inputs with shape (batch, height, width, channels) while channels\_first corresponds to inputs with shape (batch, channels, height, width). It defaults to the image\_data\_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be “channels\_last”.

# Input shape

If data\_format=’channels\_last’:

4D tensor with shape: (batch\_size, rows, cols, channels)

If data\_format=’channels\_first’:

4D tensor with shape: (batch\_size, channels, rows, cols)

# Output shape

If data\_format=’channels\_last’:

4D tensor with shape: (batch\_size, pooled\_rows, pooled\_cols, channels)

If data\_format=’channels\_first’:

4D tensor with shape: (batch\_size, channels, pooled\_rows, pooled\_cols)

5.11.Flatten

Definition: Flatten(data\_format=None, \*\*kwargs)

Type: Class in keras.layers.core module

class Flatten(Layer):

Flattens the input. Does not affect the batch size.

5.11.1.Arguments

data\_format: A string,

one of channels\_last (default) or channels\_first. The ordering of the dimensions in the inputs. The purpose of this argument is to preserve weight ordering when switching a model from one data format to another.channels\_last corresponds to inputs with shape (batch, …, channels) while channels\_first corresponds to inputs with shape (batch, channels, …). It defaults to theimage\_data\_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be “channels\_last”.

# Example

[**``**](#id1)[**`**](#id3)**python**

model = Sequential() model.add(Conv2D(64, 3, 3,

border\_mode=’same’, input\_shape=(3, 32, 32)))

# now: model.output\_shape == (None, 64, 32, 32)

model.add(Flatten()) # now: model.output\_shape == (None, 65536)

[``](#id5)[`](#id7)

5.12.Dense

Definition: Dense(self, units, activation=None, use\_bias=True, kernel\_initilizer='glorot\_uniform', bias\_initilizer='zeros', kernel\_regularizer=None, bias\_regularizer=None, activity\_regularizer=None, kernel\_constraint=None, bias\_constraint=None, \*\*kwargs)

Type: Class in keras.layers.core module

Just your regular densely-connected NN layer.

Dense implements the operation: output = activation(dot(input, kernel) + bias)where activation is the element-wise activation function passed as theactivation argument, kernel is a weights matrix created by the layer, and biasis a bias vector created by the layer (only applicable if use\_bias is True).

Note: if the input to the layer has a rank greater than 2, then it is flattened prior to the initial dot product with kernel.

# Example

[``](#id1)[`](#id3)python

# as first layer in a sequential model: model = Sequential() model.add(Dense(32, input\_shape=(16,))) # now the model will take as input arrays of shape (, 16) # and output arrays of shape (, 32)

# after the first layer, you don’t need to specify # the size of the input anymore: model.add(Dense(32))

[``](#id5)[`](#id7)

5.12.1.Arguments

units**:** Positive integer, dimensionality of the output space. activation: Activation function to use

(see [activations](../activations.md)). If you don’t specify anything, no activation is applied (ie. “linear” activation: a(x) = x).

use\_bias: Boolean, whether the layer uses a bias vector. kernel\_initializer: Initializer for the kernel weights matrix

(see [initializers](../initializers.md)).

bias\_initializer: Initializer for the bias vector

(see [initializers](../initializers.md)).

kernel\_regularizer: Regularizer function applied to

the kernel weights matrix (see [regularizer](../regularizers.md)).

bias\_regularizer**:** Regularizer function applied to the bias vector

(see [regularizer](../regularizers.md)).

activity\_regularizer: Regularizer function applied to

the output of the layer (its “activation”). (see [regularizer](../regularizers.md)).

kernel\_constraint**:** Constraint function applied to

the kernel weights matrix (see [constraints](../constraints.md)).

bias\_constraint: Constraint function applied to the bias vector

(see [constraints](../constraints.md)).

**#** Input shape

nD tensor with shape: (batch\_size, …, input\_dim). The most common situation would be a 2D input with shape (batch\_size, input\_dim).

# Output shape

nD tensor with shape: (batch\_size, …, units). For instance, for a 2D input with shape (batch\_size, input\_dim), the output would have shape (batch\_size, units).

5.13.Compile

Definition: compile(self, optimizer, loss=None, metrics=None, loss\_weighst=None, sample\_weight\_mode=None, weighted\_metrics=None, target\_tensors=None, \*\*kwargs)

Type: Method in training class in keras.engine.training module.

Configures the model for training.

5.13.1.Arguments

optimizer: String (name of optimizer) or optimizer instance.

See [optimizers](/optimizers).

loss: String (name of objective function) or objective function.

See [losses](/losses). If the model has multiple outputs, you can use a different loss on each output by passing a dictionary or a list of losses. The loss value that will be minimized by the model will then be the sum of all individual losses.

metrics: List of metrics to be evaluated by the model

during training and testing. Typically you will use metrics=[‘accuracy’]. To specify different metrics for different outputs of a multi-output model, you could also pass a dictionary, such as metrics={‘output\_a’: ‘accuracy’}.

loss\_weights: Optional list or dictionary specifying scalar

coefficients (Python floats) to weight the loss contributions of different model outputs. The loss value that will be minimized by the model will then be the weighted sum of all individual losses, weighted by the loss\_weightscoefficients. If a list, it is expected to have a 1:1 mapping to the model’s outputs. If a tensor, it is expected to map output names (strings) to scalar coefficients.

sample\_weight\_mode: If you need to do timestep-wise

sample weighting (2D weights), set this to “temporal”.None defaults to sample-wise weights (1D). If the model has multiple outputs, you can use a differentsample\_weight\_mode on each output by passing a dictionary or a list of modes.

weighted\_metrics: List of metrics to be evaluated and weighted

by sample\_weight or class\_weight during training and testing.

target\_tensors: By default, Keras will create placeholders for the

model’s target, which will be fed with the target data during training. If instead you would like to use your own target tensors (in turn, Keras will not expect external Numpy data for these targets at training time), you can specify them via the target\_tensors argument. It can be a single tensor (for a single-output model), a list of tensors, or a dict mapping output names to target tensors.

[\*\*](#id1)kwargs: When using the Theano/CNTK backends, these arguments

are passed into K.function. When using the TensorFlow backend, these arguments are passed into tf.Session.run.

5.13.2.Raises

ValueError: In case of invalid arguments for

optimizer, loss, metrics or sample\_weight\_mode.

5.14.ImageDataGenerator

Definition: ImageDataGenerator(self, featurewise\_center=False, stepwise\_center=False, featurewise\_std\_normalisation=False, stepwise\_std\_normalisation=False, zca\_whitening=False, zca\_epsilon=1e-6, rotation\_range=0., width\_shift\_range=0., height\_shift\_range=0., brightness\_range=None, shear\_range=0., zoom\_range=0., channel\_shift\_range=0., fill\_mode='nearest', cval=0., horizontal\_flip=False, vertical\_flip=False, rescale=None, prepocessing\_function=None, data\_format=None, validation\_split=0.0, \*\*kwargs)

Type: Class in keras\_prepocessing.Image module

Generate batches of tensor image data with real-time data augmentation.

The data will be looped over (in batches).

5.14.1.Arguments

featurewise\_center: Boolean.

Set input mean to 0 over the dataset, feature-wise.

samplewise\_center: Boolean. Set each sample mean to 0. featurewise\_std\_normalization: Boolean.

Divide inputs by std of the dataset, feature-wise.

samplewise\_std\_normalization: Boolean. Divide each input by its std. zca\_epsilon: epsilon for ZCA whitening. Default is 1e-6. zca\_whitening: Boolean. Apply ZCA whitening. rotation\_range: Int. Degree range for random rotations. width\_shift\_range: Float, 1-D array-like or int

float: fraction of total width, if < 1, or pixels if >= 1.

1-D array-like: random elements from the array.

int: integer number of pixels from interval

(-width\_shift\_range, +width\_shift\_range)

With width\_shift\_range=2 possible values

are integers [-1, 0, +1], same as withwidth\_shift\_range=[-1, 0, +1], while with width\_shift\_range=1.0 possible values are floats in the interval [-1.0, +1.0).

height\_shift\_range: Float, 1-D array-like or int

float: fraction of total height, if < 1, or pixels if >= 1.

1-D array-like: random elements from the array.

int: integer number of pixels from interval

(-height\_shift\_range, +height\_shift\_range)

With height\_shift\_range=2 possible values

are integers [-1, 0, +1], same as withheight\_shift\_range=[-1, 0, +1], while with height\_shift\_range=1.0 possible values are floats in the interval [-1.0, +1.0).

shear\_range: Float. Shear Intensity

(Shear angle in counter-clockwise direction in degrees)

zoom\_range: Float or [lower, upper]. Range for random zoom.

If a float, [lower, upper] = [1-zoom\_range, 1+zoom\_range].

channel\_shift\_range: Float. Range for random channel shifts. fill\_mode: One of {“constant”, “nearest”, “reflect” or “wrap”}.

Default is ‘nearest’. Points outside the boundaries of the input are filled according to the given mode: - ‘constant’: kkkkkkkk|abcd|kkkkkkkk (cval=k) - ‘nearest’: aaaaaaaa|abcd|dddddddd - ‘reflect’: abcddcba|abcd|dcbaabcd - ‘wrap’: abcdabcd|abcd|abcdabcd

cval: Float or Int.

Value used for points outside the boundaries whenfill\_mode = “constant”.

horizontal\_flip: Boolean. Randomly flip inputs horizontally. vertical\_flip: Boolean. Randomly flip inputs vertically. rescale: rescaling factor. Defaults to None.

If None or 0, no rescaling is applied, otherwise we multiply the data by the value provided (before applying any other transformation).

preprocessing\_function: function that will be implied on each input.

The function will run after the image is resized and augmented. The function should take one argument: one image (Numpy tensor with rank 3), and should output a Numpy tensor with the same shape.

data\_format: Image data format,

either “channels\_first” or “channels\_last”. “channels\_last” mode means that the images should have shape(samples, height, width, channels), “channels\_first” mode means that the images should have shape (samples, channels, height, width). It defaults to theimage\_data\_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be “channels\_last”.

validation\_split: Float. Fraction of images reserved for validation

(strictly between 0 and 1).

# Examples Example of using .flow(x, y):

[``](#id1)[`](#id3)python (x\_train, y\_train), (x\_test, y\_test) = cifar10.load\_data() y\_train = np\_utils.to\_categorical(y\_train, num\_classes) y\_test = np\_utils.to\_categorical(y\_test, num\_classes)

datagen = ImageDataGenerator(

featurewise\_center=True, featurewise\_std\_normalization=True, rotation\_range=20, width\_shift\_range=0.2, height\_shift\_range=0.2, horizontal\_flip=True)

# compute quantities required for featurewise normalization # (std, mean, and principal components if ZCA whitening is applied) datagen.fit(x\_train)

# fits the model on batches with real-time data augmentation: model.fit\_generator(datagen.flow(x\_train, y\_train, batch\_size=32),

steps\_per\_epoch=len(x\_train) / 32, epochs=epochs)

# here’s a more “manual” example for e in range(epochs):

print(‘Epoch’, e) batches = 0 for x\_batch, y\_batch in datagen.flow(x\_train, y\_train, batch\_size=32):

model.fit(x\_batch, y\_batch) batches += 1 if batches >= len(x\_train) / 32:

# we need to break the loop by hand because # the generator loops indefinitely break

[``](#id5)` Example of using .flow\_from\_directory(directory):

[``](#id7)[`](#id9)python train\_datagen = ImageDataGenerator(

rescale=1./255, shear\_range=0.2, zoom\_range=0.2, horizontal\_flip=True)

test\_datagen = ImageDataGenerator(rescale=1./255)

train\_generator = train\_datagen.flow\_from\_directory(

‘data/train’, target\_size=(150, 150), batch\_size=32, class\_mode=’binary’)

validation\_generator = test\_datagen.flow\_from\_directory(

‘data/validation’, target\_size=(150, 150), batch\_size=32, class\_mode=’binary’)

model.fit\_generator(

train\_generator, steps\_per\_epoch=2000, epochs=50, validation\_data=validation\_generator, validation\_steps=800)

[``](#id11)[`](#id13)

Example of transforming images and masks together.

[``](#id15)[`](#id17)python # we create two instances with the same arguments data\_gen\_args = dict(featurewise\_center=True,

featurewise\_std\_normalization=True, rotation\_range=90., width\_shift\_range=0.1, height\_shift\_range=0.1, zoom\_range=0.2)

image\_datagen = ImageDataGenerator([\*\*](#id19)data\_gen\_args) mask\_datagen = ImageDataGenerator([\*\*](#id21)data\_gen\_args)

# Provide the same seed and keyword arguments to the fit and flow methods seed = 1 image\_datagen.fit(images, augment=True, seed=seed) mask\_datagen.fit(masks, augment=True, seed=seed)

image\_generator = image\_datagen.flow\_from\_directory(

‘data/images’, class\_mode=None, seed=seed)

mask\_generator = mask\_datagen.flow\_from\_directory(

‘data/masks’, class\_mode=None, seed=seed)

# combine generators into one which yields image and masks train\_generator = zip(image\_generator, mask\_generator)

model.fit\_generator(

train\_generator, steps\_per\_epoch=2000, epochs=50)

[``](#id23)[`](#id25)

### 5.15. flow\_from\_directory

Definition: flow\_from\_directory(directory, target\_size=(256, 256), color\_mode='rgb', classes=None, class\_mode='categorical', batch\_size=32, shuffle=True, seed=None, save\_to\_dir=None, save\_prefix='', save\_format='png', follow\_links=False, subset=None, interpolation='nearest')

Takes the path to a directory & generates batches of augmented data.

### 5.15.1. Arguments

directory: Path to the target directory. It should contain one subdirectory per class. Any PNG, JPG, BMP, PPM or TIF images inside each of the subdirectories directory tree will be included in the generator. See [this script](https://gist.github.com/fchollet/%20%20%20%20%20%20%20%200830affa1f7f19fd47b06d4cf89ed44d) for more details.

target\_size: Tuple of integers (height, width), default: (256, 256). The dimensions to which all images found will be resized.

color\_mode: One of "grayscale", "rbg". Default: "rgb". Whether the images will be converted to have 1 or 3 color channels.

classes: Optional list of class subdirectories (e.g. ['dogs', 'cats']). Default: None. If not provided, the list of classes will be automatically inferred from the subdirectory names/structure under directory, where each subdirectory will be treated as a different class (and the order of the classes, which will map to the label indices, will be alphanumeric). The dictionary containing the mapping from class names to class indices can be obtained via the attribute class\_indices.

class\_mode: One of "categorical", "binary", "sparse", "input", or None. Default: "categorical". Determines the type of label arrays that are returned:

"categorical" will be 2D one-hot encoded labels,

"binary" will be 1D binary labels, "sparse" will be 1D integer labels,

"input" will be images identical to input images (mainly used to work with autoencoders).

If None, no labels are returned (the generator will only yield batches of image data, which is useful to use with model.predict\_generator(), model.evaluate\_generator(), etc.). Please note that in case of class\_mode None, the data still needs to reside in a subdirectory of directory for it to work correctly.

batch\_size: Size of the batches of data (default: 32).

shuffle: Whether to shuffle the data (default: True)

seed: Optional random seed for shuffling and transformations.

save\_to\_dir: None or str (default: None). This allows you to optionally specify a directory to which to save the augmented pictures being generated (useful for visualizing what you are doing).

save\_prefix: Str. Prefix to use for filenames of saved pictures (only relevant if save\_to\_dir is set).

save\_format: One of "png", "jpeg" (only relevant if save\_to\_dir is set). Default: "png".

follow\_links: Whether to follow symlinks inside class subdirectories (default: False).

subset: Subset of data ("training" or "validation") if validation\_split is set in ImageDataGenerator.

interpolation: Interpolation method used to resample the image if the target size is different from that of the loaded image. Supported methods are "nearest", "bilinear", and "bicubic". If PIL version 1.1.3 or newer is installed, "lanczos" is also supported. If PIL version 3.4.0 or newer is installed, "box" and "hamming" are also supported. By default, "nearest"is used.

### 5.15. fit\_generator

Definition**:** fit\_generator(generator, steps\_per\_epoch=None, epochs=1, verbose=1, callbacks=None, validation\_date=None, validation\_steps=None, class\_weight=None, max\_queue\_size=10, workers=1, use\_multiprocessing=False, shuffle=True, initial\_epoch=0)

Types**:** Present in keras.engine.trainig.model module

Trains the model on data generated batch-by-batch by a Python generator (or an instance of *Sequence*).

The generator is run in parallel to the model, for efficiency. For instance, this allows you to do real-time data augmentation on images on CPU in parallel to training your model on GPU.

The use of keras.utils.Sequence guarantees the ordering and guarantees the single use of every input per epoch when using use\_multiprocessing=True*.*

### 5.15.1. Arguments

generator: A generator or an instance of Sequence

(keras.utils.Sequence) object in order to avoid duplicate data when using multiprocessing. The output of the generator must be either - a tuple (inputs, targets) - a tuple (inputs, targets, sample\_weights). This tuple (a single output of the generator) makes a single batch. Therefore, all arrays in this tuple must have the same length (equal to the size of this batch). Different batches may have different sizes. For example, the last batch of the epoch is commonly smaller than the others, if the size of the dataset is not divisible by the batch size. The generator is expected to loop over its data indefinitely. An epoch finishes when steps\_per\_epoch batches have been seen by the model.

steps\_per\_epoch: Integer.

Total number of steps (batches of samples) to yield fromgenerator before declaring one epoch finished and starting the next epoch. It should typically be equal to the number of samples of your dataset divided by the batch size. Optional for Sequence: if unspecified, will use thelen(generator) as a number of steps.

epochs: Integer. Number of epochs to train the model.

An epoch is an iteration over the entire data provided, as defined by steps\_per\_epoch. Note that in conjunction withinitial\_epoch, epochs is to be understood as “final epoch”. The model is not trained for a number of iterations given by epochs, but merely until the epoch of index epochs is reached.

verbose: Integer. 0, 1, or 2. Verbosity mode.

0 = silent, 1 = progress bar, 2 = one line per epoch.

callbacks: List of keras.callbacks.Callback instances.

List of callbacks to apply during training. See [callbacks](/callbacks).

validation\_data: This can be either

a generator or a Sequence object for the validation data

tuple (x\_val, y\_val)

tuple (x\_val, y\_val, val\_sample\_weights)

on which to evaluate the loss and any model metrics at the end of each epoch. The model will not be trained on this data.

validation\_steps: Only relevant if validation\_data

is a generator. Total number of steps (batches of samples) to yield from validation\_data generator before stopping at the end of every epoch. It should typically be equal to the number of samples of your validation dataset divided by the batch size. Optional for Sequence: if unspecified, will use the len(validation\_data) as a number of steps.

class\_weight: Optional dictionary mapping class indices (integers)

to a weight (float) value, used for weighting the loss function (during training only). This can be useful to tell the model to “pay more attention” to samples from an under-represented class.

max\_queue\_size: Integer. Maximum size for the generator queue.

If unspecified, max\_queue\_size will default to 10.

workers: Integer. Maximum number of processes to spin up

when using process-based threading. If unspecified,workers will default to 1. If 0, will execute the generator on the main thread.

use\_multiprocessing: Boolean.

If True, use process-based threading. If unspecified,use\_multiprocessing will default to False. Note that because this implementation relies on multiprocessing, you should not pass non-picklable arguments to the generator as they can’t be passed easily to children processes.

shuffle: Boolean. Whether to shuffle the order of the batches at

the beginning of each epoch. Only used with instances ofSequence (keras.utils.Sequence). Has no effect whensteps\_per\_epoch is not None.

initial\_epoch: Integer.

Epoch at which to start training (useful for resuming a previous training run).

# Returns

A History object. Its History.history attribute is a record of training loss values and metrics values at successive epochs, as well as validation loss values and validation metrics values (if applicable).

# Raises

ValueError: In case the generator yields data in an invalid format.

# Example

[``](#id1)[`](#id3)python def generate\_arrays\_from\_file(path):

while True:

with open(path) as f:

for line in f:

# create numpy arrays of input data # and labels, from each line in the file x1, x2, y = process\_line(line) yield ({‘input\_1’: x1, ‘input\_2’: x2}, {‘output’: y})

model.fit\_generator(generate\_arrays\_from\_file(‘/my\_file.txt’),

steps\_per\_epoch=10000, epochs=10)

[``](#id5)[`](#id7)

### 5.16. predict\_generator

Definition**:** predict\_generator(generator, steps=None, epochs=1, verbose=0, max\_queue\_size=10, workers=1, use\_multiprocessing=False)

Types**:** Present in keras.engine.trainig.model module

Generates predictions for the input samples from a data generator.

The generator should return the same kind of data as accepted bypredict\_on\_batch.

### 5.16.1. Arguments

generator: Generator yielding batches of input samples

or an instance of Sequence (keras.utils.Sequence) object in order to avoid duplicate data when using multiprocessing.

steps: Total number of steps (batches of samples)

to yield from generator before stopping. Optional forSequence: if unspecified, will use the len(generator) as a number of steps.

max\_queue\_size: Maximum size for the generator queue. workers: Integer. Maximum number of processes to spin up

when using process based threading. If unspecified,workers will default to 1. If 0, will execute the generator on the main thread.

use\_multiprocessing: If True, use process based threading.

Note that because this implementation relies on multiprocessing, you should not pass non picklable arguments to the generator as they can’t be passed easily to children processes.

verbose: verbosity mode, 0 or 1.

5.16.2. Returns

Numpy array(s) of predictions.

5.16.3. Raises

ValueError: In case the generator yields

data in an invalid format.

**Chapter – 6**

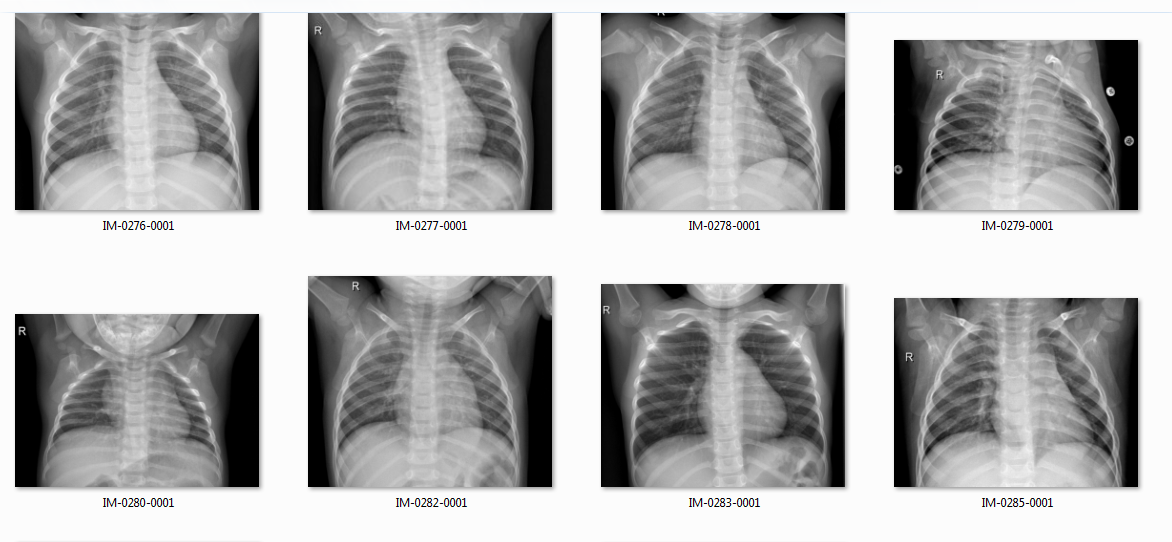
DataSet

The dataset is organized into 3 folders (train, test, val) and contains subfolders for each image category (Pneumonia/Normal). There are 5,863 X-Ray images (JPEG) and 2 categories (Pneumonia/Normal).

Chest X-ray images (anterior-posterior) were selected from retrospective cohorts of pediatric patients of one to five years old from Guangzhou Women and Children’s Medical Center, Guangzhou. All chest X-ray imaging was performed as part of patients’ routine clinical care.

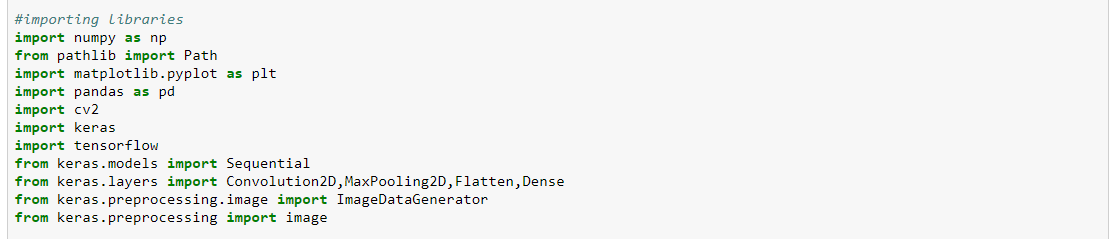
For the analysis of chest x-ray images, all chest radiographs were initially screened for quality control by removing all low quality or unreadable scans. The diagnoses for the images were then graded by two expert physicians before being cleared for training the AI system. In order to account for any grading errors, the evaluation set was also checked by a third expert.

This data was sourced from www.kaggle.com



**Chapter – 7**

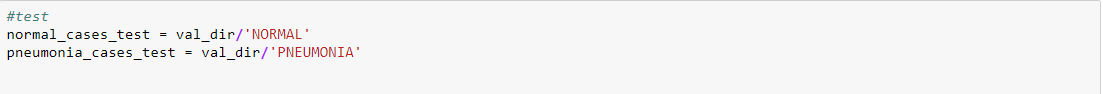
Codes Description



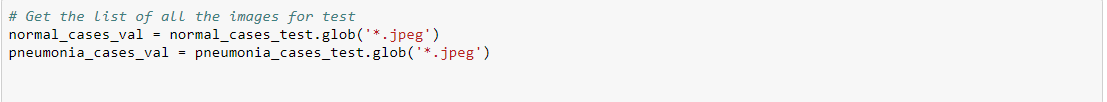
Description: Here, we are importing Libraries : Numpy, Pandas, cv2, Keras, TensorFlow. And Sub libraries from their respective libraries: Path , Pyplot, Sequential, Convolution2D, MAxPooling2D,Flatten, Dense, ImageDataGenerator, and image from pathlib, matplotlib, keras.models, keras.layers,keras.preprocessing.



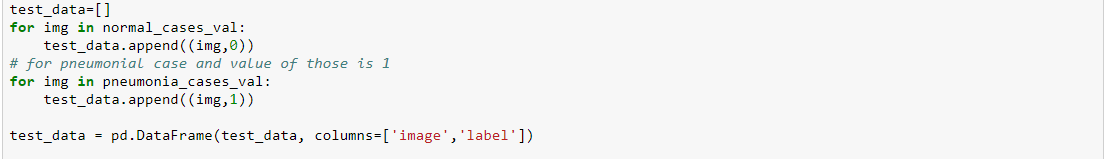
Here, we have defined path to the data directory where the images are stored. Then, we define the train directory (data for training our model) and validation directory (for validating our data).



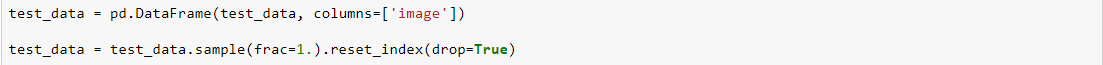
This tells how we have accessed data from the two sub-directories within the test data set which are Normal and Pneumonia.



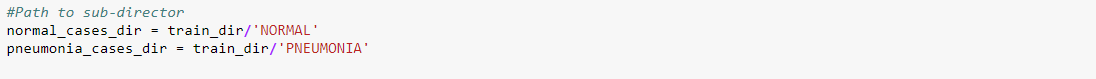
We have loaded images for validation within the two cases of normal and pneumonia.



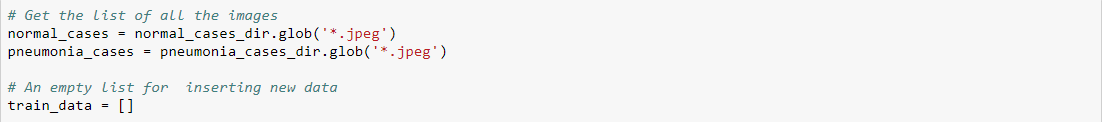
We have created a list for test data where we have appended the image and labels which define whether the lungs are infected (label 1) or not infected (label 0).



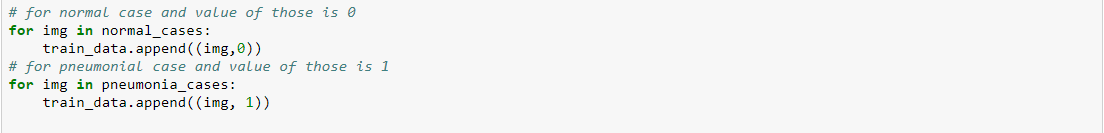
This has been used for shuffling the ‘normal’ and ‘pneumonia’ data.



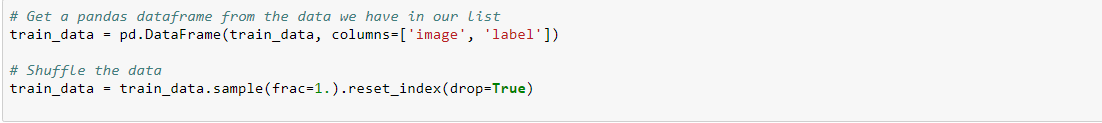
This tells how we have accessed data from the two sub-directories within the train data set which are Normal and Pneumonia.



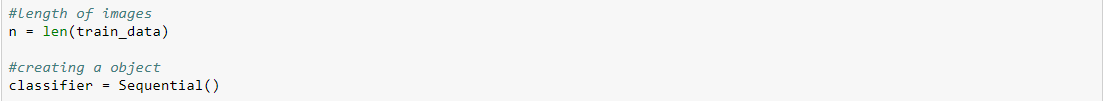
This is where we load images in normal and pneumonia cases of training dataset. And we have created a list for fitting the train data along with the labels.



We have appended image and labels according to the diagnosis to the train set. (Label 0 for normal lungs and Label 1 for infected lungs)



We have defined the data frame for train data – which will have attributes image and label. After that, we have shuffled the data for better training of model. We have reset the index to match the shuffling.



We have taken the number of data elements contained in training set in variable ‘n’. Then, we have created an object classifier of sub-library sequential which will be used to create the network, add layers, and prediction.

12.PNG

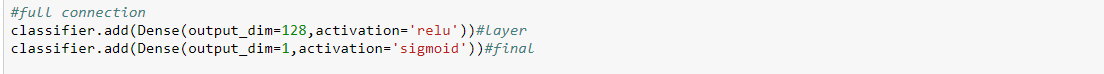
First step of creating a neural network, Convolution. The numbers of convolution matrix are 64 for each image with the dimension of 3\*3. The input image has been resized to 64\*64 with 3 signifying the matrix dimension of image. The activation function used is ‘relu’ here.

13.PNG

Second Step: Pooling. The size of pooling matrix is 2\*2 as stated above.

14.PNG

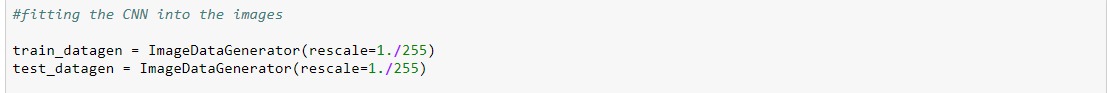
Step 3: Flatten. We have used the flatten function to convert the output into 1D vector for further processing by the last step called the dense.



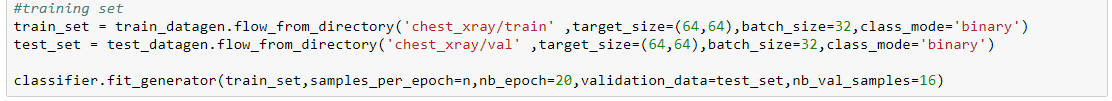
Step 4: Final step of full connection where we add multiple layers to the model ‘classifier’ for neural network. Output dimension for initial layer is 128 with activation function relu and sigmoid function for final layer.

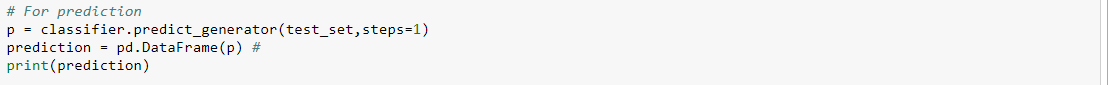
16.PNG

This displays the Compilation of the neural network with parameters optimizer , loss and metrics.



Here, we have fitted the test and train data in the Convolution Network .

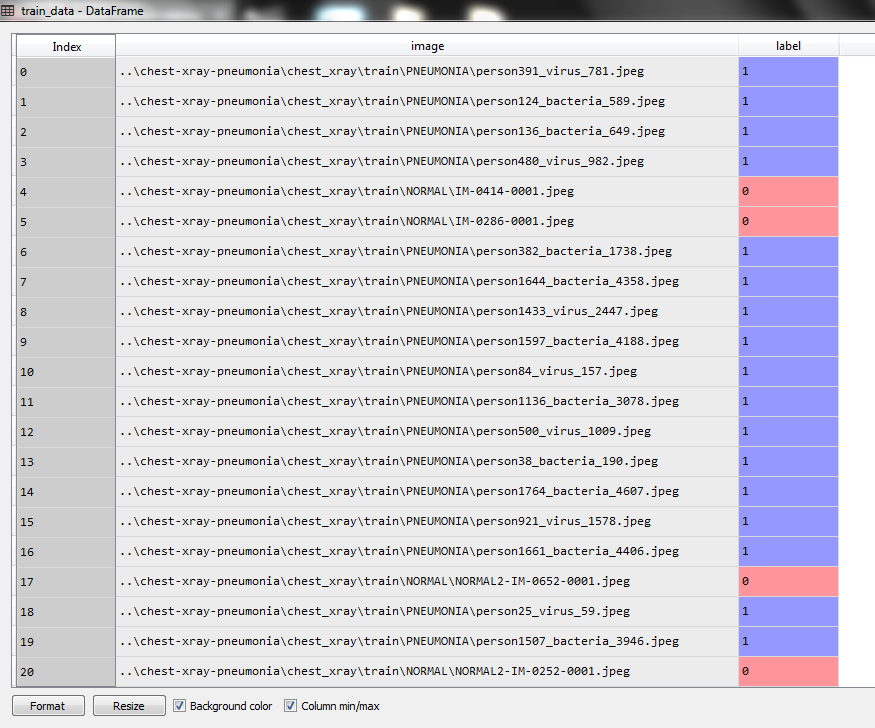
 This is the code used that trains the model according to our dataset using the CNN model that we compiled earlier.



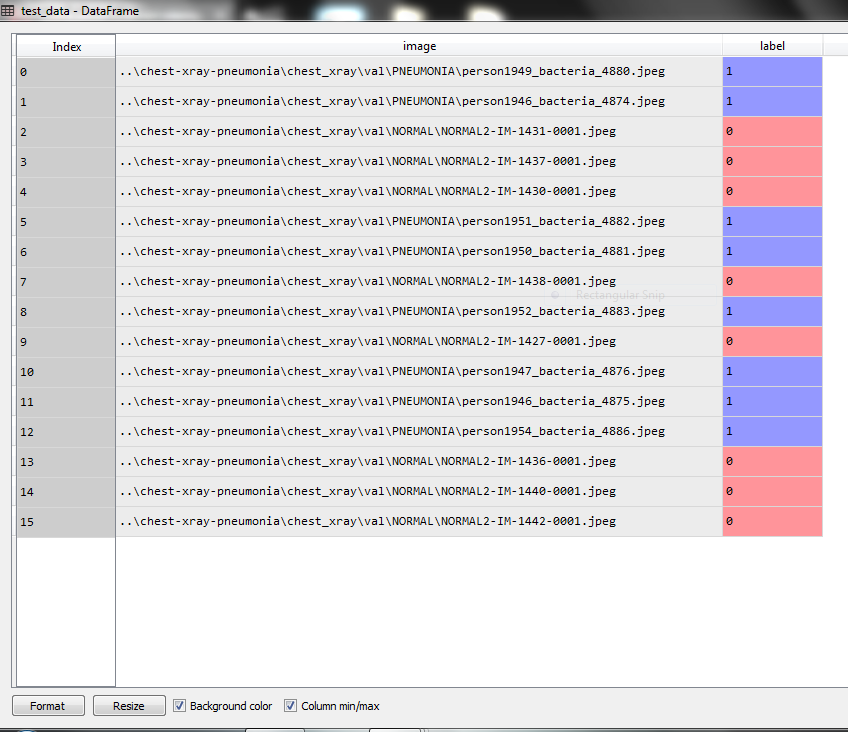
This is the part responsible for prediction where we have predicted pneumonia for the test data set and printed it the suitable dataframe.

**Chapter – 8**

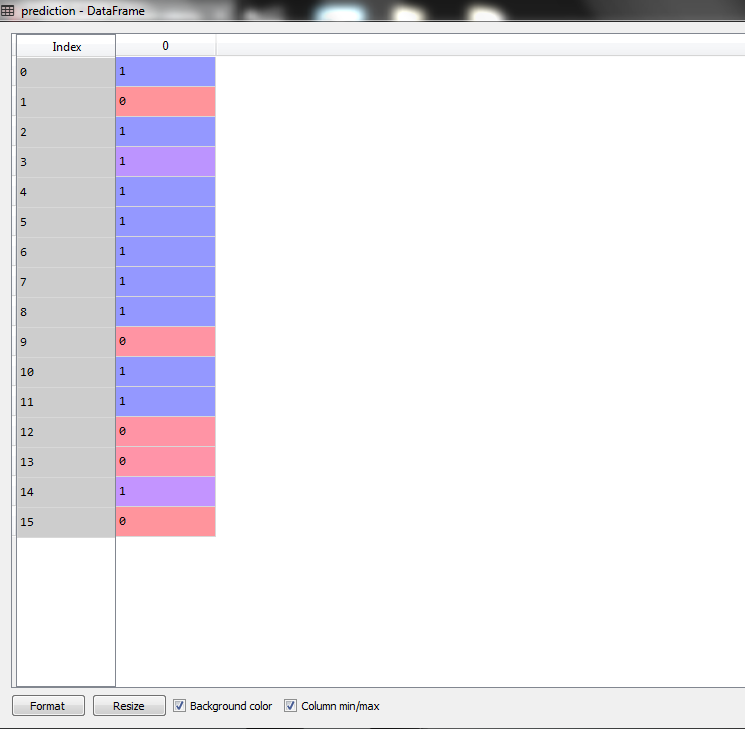
Variable Explorer



(Training Data)



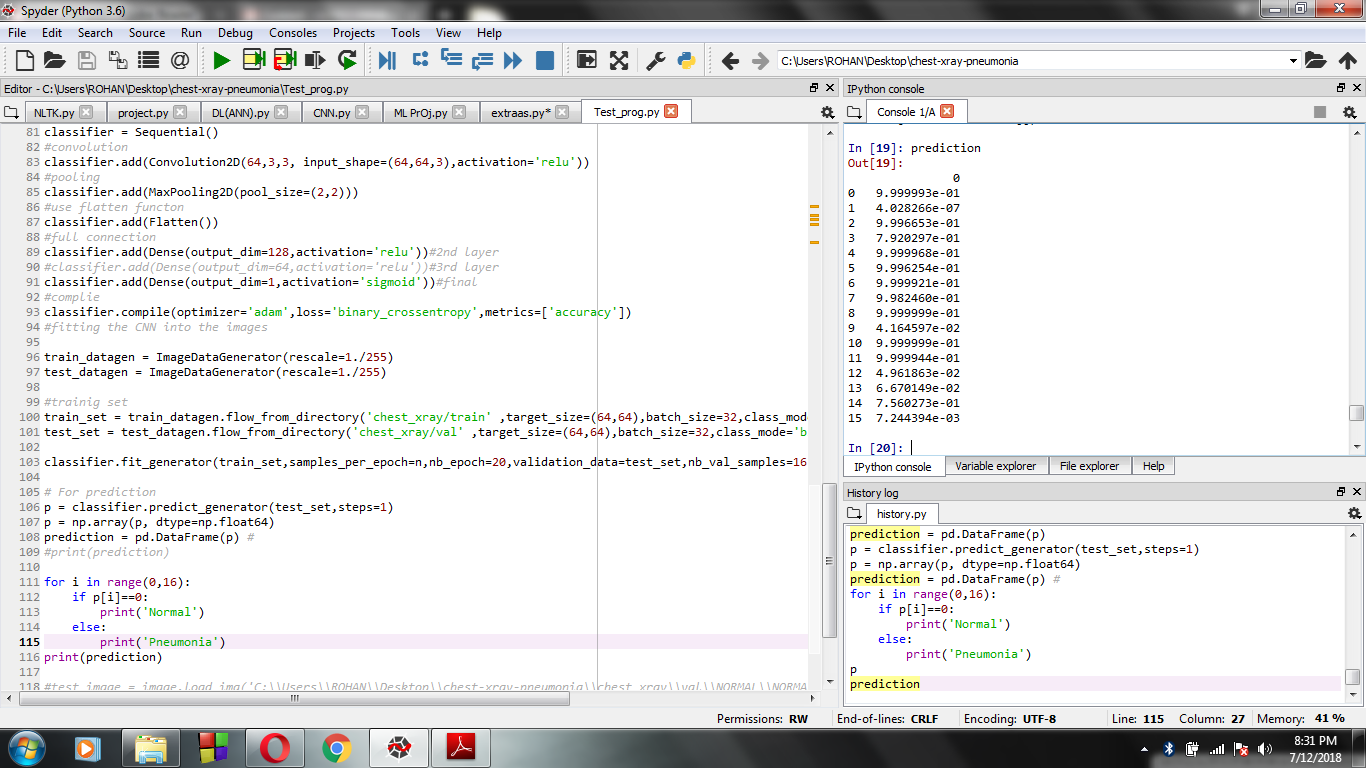
(Test Data)



(Prediction Array)

**Chapter – 9**

IPython Console



**Chapter – 10**

Code

#importing libraries

import numpy as np

from pathlib import Path

import matplotlib.pyplot as plt

import pandas as pd

import cv2

import keras

import tensorflow

from keras.models import Sequential

from keras.layers import Convolution2D,MaxPooling2D,Flatten,Dense

from keras.preprocessing.image import ImageDataGenerator

from keras.preprocessing import image

# Define path to the data directory

data\_dir = Path('..\chest-xray-pneumonia\chest\_xray')

# Path to train directory (Fancy pathlib...no more os.path!!)

train\_dir = data\_dir/'train'

# Path to validation directory

val\_dir = data\_dir/'val'

#test

normal\_cases\_test = val\_dir/'NORMAL'

pneumonia\_cases\_test = val\_dir/'PNEUMONIA'

# Get the list of all the images for test

normal\_cases\_val = normal\_cases\_test.glob('\*.jpeg')

pneumonia\_cases\_val = pneumonia\_cases\_test.glob('\*.jpeg')

test\_data=[]

for img in normal\_cases\_val:

test\_data.append((img,0))

# for pneumonial case and value of those is 1

for img in pneumonia\_cases\_val:

test\_data.append((img,1))

test\_data = pd.DataFrame(test\_data, columns=['image','label'])

test\_data = test\_data.sample(frac=1.).reset\_index(drop=True)

#Path to sub-director

normal\_cases\_dir = train\_dir/'NORMAL'

pneumonia\_cases\_dir = train\_dir/'PNEUMONIA'

# Get the list of all the images

normal\_cases = normal\_cases\_dir.glob('\*.jpeg')

pneumonia\_cases = pneumonia\_cases\_dir.glob('\*.jpeg')

# An empty list for inserting new data

train\_data = []

# for normal case and value of those is 0

for img in normal\_cases:

train\_data.append((img,0))

# for pneumonial case and value of those is 1

for img in pneumonia\_cases:

train\_data.append((img, 1))

# Get a pandas dataframe from the data we have in our list

train\_data = pd.DataFrame(train\_data, columns=['image', 'label'])

# Shuffle the data

train\_data = train\_data.sample(frac=1.).reset\_index(drop=True)

#length of images

n = len(train\_data)

#creating a object

classifier = Sequential()

#convolution

classifier.add(Convolution2D(64,3,3, input\_shape=(64,64,3),activation='relu'))

#pooling

classifier.add(MaxPooling2D(pool\_size=(2,2)))

#use flatten functon

classifier.add(Flatten())

#full connection

classifier.add(Dense(output\_dim=128,activation='relu'))#2nd layer

#classifier.add(Dense(output\_dim=64,activation='relu'))#3rd layer

classifier.add(Dense(output\_dim=1,activation='sigmoid'))#final

#complie

classifier.compile(optimizer='adam',loss='binary\_crossentropy',metrics=['accuracy'])

#fitting the CNN into the images

train\_datagen = ImageDataGenerator(rescale=1./255)

test\_datagen = ImageDataGenerator(rescale=1./255)

#trainig set

train\_set=train\_datagen.flow\_from\_directory('chest\_xray/train' ,target\_size=(64,64),batch\_size=32,class\_mode='binary')

test\_set=test\_datagen.flow\_from\_directory('chest\_xray/val' ,target\_size=(64,64),batch\_size=32,class\_mode='binary')

classifier.fit\_generator(train\_set,samples\_per\_epoch=n,nb\_epoch=20,validation\_data=test\_set,nb\_val\_samples=16)

# For prediction

p = classifier.predict\_generator(test\_set,steps=1)

prediction = pd.DataFrame(p) #

print(prediction)

**Chapter – 11**

Summary

We conclude from the project that we were successfully able to make a Pneumonia Diagnosis Prediction System. When we input images into the system, the algorithm uses the various Machine learning and Deep Learning concepts such as Convolution Neural Network to help predict whether a given pair of lungs are infected with Pneumonia or not. Also, if infected, it tells if the infection is bacterial or viral.

The system that we built has an accuracy of 50-60%. Currently we are further building on this algorithm and trying to improve its accuracy. We also plan to expand our project in the future so that it can be used to predict and diagnose any type of disease, not just pneumonia.