**1.** Study of Deep learning Packages: Tensorflow, Keras, Theano and PyTorch. Document the distinct

features and functionality of the packages.

**2.** Implementing Feedforward neural networks with Keras and TensorFlow

a. Import the necessary packages

b. Load the training and testing data (MNIST/CIFAR10)

c. Define the network architecture using Keras

d. Train the model using SGD

e. Evaluate the network

f. Plot the training loss and accuracy

**3.** Build the Image classification model by dividing the model into following 4 stages:

a. Loading and preprocessing the image data

b. Defining the model’s architecture

c. Training the model

d. Estimating the model’s performance

**4.** Use Autoencoder to implement anomaly detection. Build the model by using:

a. Import required libraries

b. Upload / access the dataset

c. Encoder converts it into latent representation

d. Decoder networks convert it back to the original input

e. Compile the models with Optimizer, Loss, and Evaluation Metrics

**5**. Implement the Continuous Bag of Words (CBOW) Model. Stages can be:

a. Data preparation

b. Generate training data

c. Train model

d. Output

**6.** Object detection using Transfer Learning of CNN architectures

**Pip install “tensorflow<3.0”**

**Import tensorflow as tf**

**1st Algorithm**

TensorFlow

Google’s Brain team developed a Deep Learning Framework called TensorFlow,

which supports languages like Python and R, and uses dataflow graphs to process data.

This is very important because as you build these neural networks, you can look at how

the data flows through the neural network.

TensorFlow’s machine learning models are easy to build, can be used for robust

machine learning production, and allow powerful experimentation for research.

With TensorFlow, you also get TensorBoard for data visualization, which is a large

package that generally goes unnoticed. TensorBoard simplifies the process for visually

displaying data when working with your shareholders. You can use the R and Python

visualization packages as well.

Keras

Francois Chollet originally developed Keras, with 350,000+ users and 700+ open-

source contributors, making it one of the fastest-growing deep learning framework

packages.

Keras supports high-level neural network API, written in Python. What makes Keras

interesting is that it runs on top of TensorFlow, Theano, and CNTK.

Keras is used in several startups, research labs, and companies including Microsoft

Research, NASA, Netflix, and Cern.

Other Features of Keras:

 User-friendly, as it offers simple APIs and provides clear and actionable feedback

upon user error

 Provides modularity as a sequence or a graph of standalone, fully-configurable

modules that can be combined with as few restrictions as possible

 Easily extensible as new modules are simple to add, making Keras suitable for

advanced research

PyTorch

Adam Paszke, Sam Gross, Soumith Chintala, and Gregory Chanan

authored PyTorch and is primarily developed by Facebook's AI Research lab (FAIR).

It’s built on the Lua-based scientific computing framework for machine learning and

deep learning algorithms. PyTorch employed Python, CUDA, along with C/C++

libraries, for processing and was designed to scale the production of building models

and overall flexibility. If you’re well-versed with C/C++, then PyTorch might not be

too big of a jump for you.

PyTorch is widely used in large companies like Facebook, Twitter, and Google.

Other Features of the Deep Learning Framework Include:

 It provides flexibility and speed due to its hybrid front-end.

 Enables scalable distributed training and performance optimization in research and

production using the “torch distributed” backend.

 Deep integration with Python allows popular libraries and packages to be quickly

write neural network layers in Python.

Theano

The University de Montreal developed Theano, written in Python and centers around

NVIDIA CUDA, allowing users to integrate it with GPS. The Python library allows

users to define, optimize, and evaluate mathematical expressions involving multi-

dimensional arrays.

**2nd Algorithm**

 Steps:

1) install python

2) install anaconda navigator

3) launch jupyter notebook

4) install keras, numpy, matplotlib and tensorflow

5) from matplotlib import pyplot

6) write the code in jupyter notebook.

7)program will execute.

**3rd Algorithm**

1. Choose a dataset of your interest or you can also create your own image dataset

(Ref : https://www.kaggle.com/datasets/) Import all necessary files.

( Ref : https://www.analyticsvidhya.com/blog/2021/01/image-classification-using-convolutional-

neural-networks-a-step-by-step-guide/)

Libraries and functions required

1. Tensorflow,keras

numpy : NumPy is a Python library used for working with arrays. It also has functions for

working in domain of linear algebra, fourier transform, and matrices. NumPy stands for

Numerical Python. To import numpy use

import numpy as np

pandas: pandas is a fast, powerful, flexible and easy to use open source data analysis and

manipulation tool, built on top of the Python programming language. To import pandas use

import pandas as pd

sklearn : Scikit-learn (Sklearn) is the most useful and robust library for machine learning in

Python. It provides a selection of efficient tools for machine learning and statistical modeling

including classification, regression, clustering and dimensionality reduction via a consistence

interface in Python. This library, which is largely written in Python, is built upon NumPy, SciPy

and Matplotlib. For importing train\_test\_ split use

2. Prepare Dataset for Training : //Preparing our dataset for training will

involve assigning pathsand creating categories(labels), resizing our images.

3. Create a Training a Data : // Training is an array that will contain image

pixel values and theindex at which the image in the CATEGORIES list.

4. Shuffle the Dataset

5. Assigning Labels and Features

6. Normalising X and converting labels to categorical data

7. Split X and Y for use in CNN

8. Define, compile and train the CNN Model

9. Accuracy and Score of model.

**4th Algorithm**

Steps/ Algorithm

1. Dataset link and libraries :

Dataset : http://storage.googleapis.com/download.tensorflow.org/data/ecg.csv

Libraries required :

Pandas and Numpy for data manipulation

Tensorflow/Keras for Neural Networks

Scikit-learn library for splitting the data into train-test samples, and for some basic model

evaluation

For Model building and evaluation following libraries:

sklearn.metrics import accuracy\_score

tensorflow.keras.optimizers import Adam

sklearn.preprocessing import MinMaxScaler

tensorflow.keras import Model, Sequential

tensorflow.keras.layers import Dense, Dropout

tensorflow.keras.losses import MeanSquaredLogarithmicError

Ref:https://www.analyticsvidhya.com/blog/2021/05/anomaly-detection-using-autoencoders-a-

walk-through-in-python/

a) Import following libraries from SKlearn : i) MinMaxscaler (sklearn.preprocessing) ii)

Accuracy(sklearn.metrics) . iii) train\_test\_split (model\_selection)

b) Import Following libraries from tensorflow.keras : models , layers,optimizers,datasets ,

and set to respective values.

c) Grab to ECG.csv required dataset

d) Find shape of dataset

e) Use train\_test\_split from sklearn to build model (e.g. train\_test\_split(

features, target, test\_size=0.2, stratify=target)

f) Take usecase Novelty detection hence select training data set as Target class is 1 i.e.

Normal class

g) Scale the data using MinMaxScaler.

h) Create Autoencoder Subclass by extending model class from keras.

i) Select parameters as i)Encoder : 4 layers ii) Decoder : 4 layers iii) Activation Function :

Relu iv) Model : sequential.

j) Configure model with following parametrs : epoch = 20 , batch size =512 and compile

with Mean Squared Logarithmic loss and Adam optimizer.

e.g. model = AutoEncoder(output\_units=x\_train\_scaled.shape[1])

# configurations of model

model.compile(loss='msle', metrics=['mse'], optimizer='adam')

history = model.fit(

x\_train\_scaled,

x\_train\_scaled,

epochs=20,

batch\_size=512,

validation\_data=(x\_test\_scaled, x\_test\_scaled)

k) Plot loss,Val\_loss, Epochs and msle loss

l) Find threshold for anomaly and do predictions :

e.g. : find\_threshold(model, x\_train\_scaled):

reconstructions = model.predict(x\_train\_scaled)

# provides losses of individual instances

reconstruction\_errors = tf.keras.losses.msle(reconstructions, x\_train\_scaled)

# threshold for anomaly scores

threshold = np.mean(reconstruction\_errors.numpy()) \

+ np.std(reconstruction\_errors.numpy())

return threshold

m) Get accuracy score

**5th Algorithm**

1. Dataset link and libraries :

Create any English 5 to 10 sententece paragraph as input

Import following data from keras :

keras.models import Sequential

keras.layers import Dense, Embedding, Lambda

keras.utils import np\_utils

keras.preprocessing import sequence

keras.preprocessing.text import Tokenizer

Import Gensim for NLP operations : requirements :

Gensim runs on Linux, Windows and Mac OS X, and should run on any other platform that

supports Python 3.6+ and NumPy. Gensim depends on the following software: Python, tested

with versions 3.6, 3.7 and 3.8. NumPy for number crunching.

Ref: https://analyticsindiamag.com/the-continuous-bag-of-words-cbow-model-in-nlp-hands-on-

implementation-with-codes/

a) Import following libraries gemsim and numpy set i.e. text file created . It should be

preprocessed.

b) Tokenize the every word from the paragraph . You can call in built tokenizer present in

Gensim

c) Fit the data to tokenizer

d) Find total no of words and total no of sentences.

e) Generate the pairs of Context words and target words :

e.g. cbow\_model(data, window\_size, total\_vocab):

total\_length = window\_size\*2

for text in data:

text\_len = len(text)

for idx, word in enumerate(text):

context\_word = []

target = []

begin = idx - window\_size

end = idx + window\_size + 1

context\_word.append([text[i] for i in range(begin, end) if 0 <= i < text\_len and i

!= idx])

target.append(word)

contextual = sequence.pad\_sequences(context\_word, total\_length=total\_length)

final\_target = np\_utils.to\_categorical(target, total\_vocab)

yield(contextual, final\_target)

f) Create Neural Network model with following parameters . Model type : sequential

Layers : Dense , Lambda , embedding. Compile Options :

(loss='categorical\_crossentropy', optimizer='adam')

g) Create vector file of some word for testing

e.g.:dimensions=100

vect\_file = open('/content/gdrive/My Drive/vectors.txt' ,'w')

vect\_file.write('{} {}\n'.format(total\_vocab,dimensions)

h) Assign weights to your trained model

e.g. weights = model.get\_weights()[0]

for text, i in vectorize.word\_index.items():

final\_vec = ' '.join(map(str, list(weights[i, :])))

vect\_file.write('{} {}\n'.format(text, final\_vec)

Close()

i) Use the vectors created in Gemsim :

e.g. cbow\_output =

gensim.models.KeyedVectors.load\_word2vec\_format('/content/gdrive/

My Drive/vectors.txt', binary=False)

j) choose the word to get

similar type of words:

cbow\_output.most\_similar(posi

tive=['Your word'])

**6th Algorithm**

Steps/ Algorithm

1. Dataset link and libraries :

https://data.caltech.edu/records/mzrjq-6wc02

separate the data into training, validation, and testing sets with a 50%, 25%, 25% split and

then structured the directories as follows:

/datadir

/train

/class1

/class2

.

.

/valid

/class1

/class2

.

.

/test

/class1

/class2

.

Libraries required :

PyTorch

torchvision import transforms

torchvision import d

atasets

torch.utils.data import DataLoader

torchvision import models

torch.nn as nn

torch import optim

Ref: https://towardsdatascience.com/transfer-learning-with-convolutional-neural-networks-in-

pytorch-dd09190245ce

m) Prepare the dataset in splitting in three directories Train , alidation and test with 50 25 25

n) Do pre-processing on data with transform from Pytorch

Training dataset transformation as follows :

transforms.Compose([

transforms.RandomResizedCrop(size=256, scale=(0.8, 1.0)),

transforms.RandomRotation(degrees=15),

transforms.ColorJitter(),

transforms.RandomHorizontalFlip(),

transforms.CenterCrop(size=224), # Image net standards

transforms.ToTensor(),

transforms.Normalize([0.485, 0.456, 0.406],

[0.229, 0.224, 0.225]) # Imagenet standards

Validation Dataset transform as follows :

transforms.Compose([

transforms.Resize(size=256),

transforms.CenterCrop(size=224),

transforms.ToTensor(),

transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

o) Create Datasets and Loaders :

data = {

'train':(Our name given to train data set dir created )

datasets.ImageFolder(root=traindir, transform=image\_transforms['train']),

'valid':

datasets.ImageFolder(root=validdir, transform=image\_transforms['valid']),

}

dataloaders = {

'train': DataLoader(data['train'], batch\_size=batch\_size, shuffle=True),

'val': DataLoader(data['valid'], batch\_size=batch\_size, shuffle=True)

}

p) Load Pretrain Model : from torchvision import models

model = model.vgg16(pretrained=True)

q) Freez all the Models Weight

for param in model.parameters():

param.requires\_grad = False

r) Add our own custom classifier with following parameters :

Fully connected with ReLU activation, shape = (n\_inputs, 256)

Dropout with 40% chance of dropping

Fully connected with log softmax output, shape = (256, n\_classes)

import torch.nn as nn

# Add on classifier

model.classifier[6] = nn.Sequential(

nn.Linear(n\_inputs, 256),

nn.ReLU(),

nn.Dropout(0.4),

nn.Linear(256, n\_classes),

nn.LogSoftmax(dim=1))

s) Only train the sixth layer of classifier keep remaining layers off .

Sequential(

(0): Linear(in\_features=25088, out\_features=4096, bias=True)

(1): ReLU(inplace)

(2): Dropout(p=0.5)

(3): Linear(in\_features=4096, out\_features=4096, bias=True)

(4): ReLU(inplace)

(5): Dropout(p=0.5)

(6): Sequential(

(0): Linear(in\_features=4096, out\_features=256, bias=True)

(1): ReLU()

(2): Dropout(p=0.4)

(3): Linear(in\_features=256, out\_features=100, bias=True)

(4): LogSoftmax()

)

)

t) Initialize the loss and optimizer

criteration = nn.NLLLoss()

optimizer = optim.Adam(model.parameters())

u) Train the model using Pytorch

for epoch in range(n\_epochs):

for data, targets in trainloader:

# Generate predictions

out = model(data)

# Calculate loss

loss = criterion(out, targets)

# Backpropagation

loss.backward()

# Update model parameters

optimizer.step()

v) Perform Early stopping

w) Draw performance curve

x) Calculate Accuracy

pred = torch.max(ps, dim=1)

equals = pred == targets

# Calculate accuracy

accuracy = torch.mean(equals)