ASSIGNMENT NO 01

❖ Aim

study of deep learning packages: tensorflow, keras, theano and pytorch. document the distinct features and functionality of the packages.

❖ Objective(s)

To study Deep Learning Packages, features and functionality.

❖ Scope

Enables scalable distributed training and performance optimization in research and production and also gives faster execution.

❖ Theory

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.

Initial release:	November 9, 2015
Stable release:	2.4.1 / January 21, 2021
Written in:	Python, C++, CUDA
Platform:	Linux, macOS, Windows, Android, JavaScript
Type:	Machine learning library
Repository	github.com/tensorflow/tensorflow
License:	Apache License 2.0
Website	www.tensorflow.org

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

Initial release:	March 27, 2015
Stable release:	2.4.0 / June 17, 2020
Platform:	Cross-platform
Type:	Neural networks
Repository	github.com/keras-team/keras
License:	Massachusetts Institute of Technology (MIT)
Website	https://keras.io/

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.

Initial release:	September 2016
Stable release:	1.7.1 / December 10, 2020
Platform:	IA-32, x86-64
Type:	Library for machine learning and deep learning
Repository	github.com/pytorch/pytorch
License:	Berkeley Software Distribution (BSD)
Website	https://pytorch.org/

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.

Initial release:	2007
Stable release:	1.0.5 / July 27, 2020
Platform:	Linux, macOS, Windows
Type:	Machine learning library
Repository	github.com/pytorch/pytorch
License:	The 3-Clause Berkeley Software Distribution (BSD)
Website	http://www.deeplearning.net/software/theano/

❖ Conclusion

In this practical we study the packages of deep learning like tensorflow, keras, Theano, PyTorch.

.

ASSIGNMENT NO 02

- **Aim**

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

- **Objective(s)**

To evaluate the feed forward neural Network by loading and training the data.

- **System Used: windows 10**

- **Software used : Anaconda, python 3.10**

- **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.

Conclusion:

In such a way we evaluate the network and also train and load the data.

```
import tensorflow as tf
from keras.models import Sequential
from·keras.datasets·import·mnist
import·matplotlib.pyplot·as·plt
import·numpy·as·np
import random
```

```
(x_train,y_train),(x_test,y_test)=mnist.load_data()
x_train=x_train/255
x_test=x_test/255
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mn
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
```

```
import keras
model=Sequential()
model.add(keras.layers.Flatten(input_shape=(28,28)))
model.add(keras.layers.Dense(128,activation='relu'))
model.add(keras.layers.Dense(10,activation='softmax'))
```

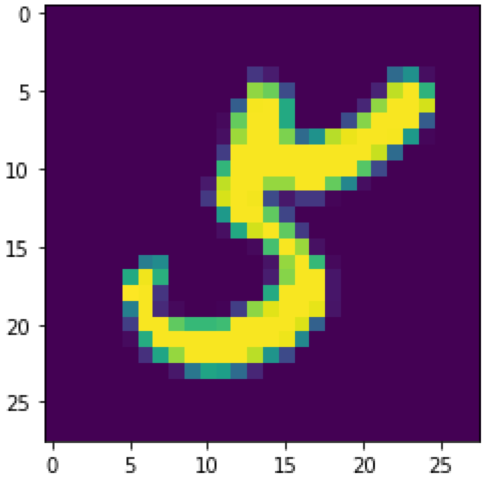
```
model.compile(optimizer='sgd',loss='sparse_categorical_crossentropy',
metrics=["accuracy"])
```

```
H=model.fit(x_train,y_train,validation_data=(x_test,y_test),epochs=5)
```

```
Epoch 1/5
1875/1875 [==============================] - 5s 2ms/step - loss: 0.6389 - accuracy:
Epoch 2/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.3357 - accuracy:
Epoch 3/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2880 - accuracy:
Epoch 4/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2581 - accuracy:
Epoch 5/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2357 - accuracy:
```

```
test_loss, test_acc = model.evaluate(x_test,y_test)
print("Loss = %.3f"%test_loss)
print("Accuracy = %.3f"%test_acc)
n=random.randint(0,9999)
plt.imshow(x_test[n])
plt.show()
prediction=model.predict(x_test)
print("The handwritten number in the image is %d" % np.argmax(prediction[n]))
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.2198 - accuracy:
Loss = 0.220
Accuracy = 0.938
```



```
The handwritten number in the image is 5
```

Colab paid products - Cancel contracts here

Assignment No.3

Title : Build the Image classification model

Aim: Build the Image classification model by dividing the model into following 4 stages:

a. Loading and pre-processing the image data

b. Defining the model's architecture

c. Training the model

d. Estimating the model's performance

Theory : 1) What is Image classification problem?
2) Why to use Deep learning for Image classification ? State and compare different Type of Neural Networks used for the Image classification?
3) What is CNN?
4) Explain Convolution operation and Convolution kernel related to Deep learning.
5) Explain how kernel operate on the Input image by taking sample matrix.
6) Explain the types of convolution and convolution layers related to CNN.
7) Explain how the feature extraction is done with convolution layers?

Steps/ 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.

Sample Code with comments and Output : Attach Printout with Output .

Conclusion :

As per the evalution of model write down in line with your output about accuracy and otherevaluation parameters.

```
!pip install PyDrive
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/
Requirement already satisfied: PyDrive in /usr/local/lib/python3.7/dist-packages (1.
Requirement already satisfied: oauth2client>=4.0.0 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: PyYAML>=3.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: google-api-python-client>=1.2 in /usr/local/lib/pytho
Requirement already satisfied: google-api-core<3dev,>=1.21.0 in /usr/local/lib/pytho
Requirement already satisfied: httplib2<1dev,>=0.15.0 in /usr/local/lib/python3.7/di
Requirement already satisfied: six<2dev,>=1.13.0 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: google-auth<3dev,>=1.16.0 in /usr/local/lib/python3.7
Requirement already satisfied: google-auth-httplib2>=0.0.3 in /usr/local/lib/python3
Requirement already satisfied: uritemplate<4dev,>=3.0.0 in /usr/local/lib/python3.7/
Requirement already satisfied: setuptools>=40.3.0 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: pytz in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: requests<3.0.0dev,>=2.18.0 in /usr/local/lib/python3.
Requirement already satisfied: protobuf<4.0.0dev,>=3.12.0 in /usr/local/lib/python3.
Requirement already satisfied: googleapis-common-protos<2.0dev,>=1.6.0 in /usr/local
Requirement already satisfied: packaging>=14.3 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: cachetools<5.0,>=2.0.0 in /usr/local/lib/python3.7/di
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.7/dis
Requirement already satisfied: pyasn1>=0.1.7 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.7/
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-package
```

```
import os
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
```

```
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

```
download = drive.CreateFile({'id': '1RmeW7TICbADS9zhu74qcwfiIX7IyjIkT'})
```

```
download.GetContentFile('train_LbELtWX.zip')
!unzip train_LbELtWX.zip
```

```
 extracting: train/12189.png
 extracting: train/59682.png
 extracting: train/27484.png
 extracting: train/13061.png
 extracting: train/8508.png
 extracting: train/33908.png
 extracting: train/13013.png
 extracting: train/29204.png
```

```
  extracting: train/45944.png
  extracting: train/57295.png
  extracting: train/39834.png
  extracting: train/47060.png
  extracting: train/33793.png
  extracting: train/31482.png
   inflating: train/45449.png
  extracting: train/3954.png
  extracting: train/29685.png
  extracting: train/42965.png
  extracting: train/56719.png
  extracting: train/3816.png
  extracting: train/46664.png
  extracting: train/55244.png
  extracting: train/8436.png
  extracting: train/39121.png
  extracting: train/45727.png
  extracting: train/57618.png
  extracting: train/36753.png
  extracting: train/8505.png
  extracting: train/59084.png
  extracting: train/8094.png
  extracting: train/41350.png
  extracting: train/30092.png
  extracting: train/59750.png
  extracting: train/27454.png
  extracting: train/18501.png
  extracting: train/25421.png
  extracting: train/44863.png
  extracting: train/42580.png
  extracting: train/30193.png
  extracting: train/26105.png
  extracting: train/44744.png
  extracting: train/12593.png
  extracting: train/26822.png
  extracting: train/27631.png
  extracting: train/50584.png
  extracting: train/8665.png
  extracting: train/24658.png
  extracting: train/5361.png
  extracting: train/27807.png
  extracting: train/27839.png
  extracting: train/30137.png
  extracting: train/3951.png
  extracting: train/54670.png
  extracting: train/25883.png
  extracting: train/33779.png
  extracting: train/47215.png
  extracting: train/51111.png
   inflating: train.csv
```

```
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.utils import to_categorical
from keras.preprocessing import image
import numpy as np
```

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
from tqdm import tqdm
from tensorflow.keras.preprocessing import image
```

```
train = pd.read_csv('train.csv')
```

```
# We have grayscale images, so while loading the images we will keep grayscale=True, if yo
train_image = []
for i in tqdm(range(train.shape[0])):
    img = image.load_img('train/'+train['id'][i].astype('str')+'.png', target_size=(28,28,
    img = image.img_to_array(img)
    img = img/255
    train_image.append(img)
X = np.array(train_image)
```

```
100%|██████████| 60000/60000 [00:11<00:00, 5264.10it/s]
```

```
y=train['label'].values
y = to_categorical(y)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42, test_size=0.2)
```

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),activation='relu',input_shape=(28,28,1)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
```

```
model.compile(loss='categorical_crossentropy',optimizer='Adam',metrics=['accuracy'])
```

```
model.fit(X_train, y_train, epochs=10, validation_data=(X_test, y_test))
```

```
Epoch 1/10
1500/1500 [==============================] - 102s 68ms/step - loss: 0.4983 - accurac
Epoch 2/10
1500/1500 [==============================] - 99s 66ms/step - loss: 0.3288 - accuracy
Epoch 3/10
1500/1500 [==============================] - 100s 66ms/step - loss: 0.2794 - accurac
Epoch 4/10
1500/1500 [==============================] - 98s 66ms/step - loss: 0.2512 - accuracy
Epoch 5/10
1500/1500 [==============================] - 99s 66ms/step - loss: 0.2196 - accuracy
Epoch 6/10
1500/1500 [==============================] - 98s 66ms/step - loss: 0.2048 - accuracy
Epoch 7/10
```

```
1500/1500 [==============================] - 99s 66ms/step - loss: 0.1869 - accuracy
Epoch 8/10
1500/1500 [==============================] - 98s 65ms/step - loss: 0.1734 - accuracy
Epoch 9/10
1500/1500 [==============================] - 100s 66ms/step - loss: 0.1640 - accurac
Epoch 10/10
1500/1500 [==============================] - 97s 65ms/step - loss: 0.1509 - accuracy
<keras.callbacks.History at 0x7f39a122a950>
```

```
download = drive.CreateFile({'id': '1tnT4Sw5uwXnkzqFZ7LbpG1c9FkTtKw63'})
download.GetContentFile('test_ScVgIM0.zip')
!unzip test_ScVgIM0.zip
```

```
 extracting: test/67681.png
 extracting: test/65324.png
 extracting: test/64647.png
 extracting: test/63161.png
 extracting: test/69388.png
 extracting: test/68799.png
 extracting: test/68822.png
 extracting: test/63444.png
 extracting: test/68141.png
 extracting: test/62219.png
 extracting: test/65750.png
 extracting: test/61531.png
 extracting: test/63960.png
 extracting: test/61529.png
 extracting: test/64951.png
 extracting: test/68693.png
 extracting: test/69238.png
 extracting: test/60991.png
 extracting: test/67448.png
 extracting: test/62909.png
 extracting: test/61994.png
 extracting: test/68284.png
 extracting: test/67792.png
 extracting: test/64362.png
 extracting: test/61191.png
 extracting: test/62576.png
 extracting: test/60708.png
 extracting: test/66440.png
 extracting: test/62156.png
 extracting: test/68481.png
 extracting: test/67597.png
 extracting: test/67598.png
 extracting: test/65897.png
 extracting: test/65208.png
 extracting: test/60053.png
 extracting: test/68238.png
 extracting: test/64976.png
 extracting: test/63555.png
 extracting: test/62716.png
 extracting: test/66290.png
 extracting: test/64457.png
 extracting: test/62412.png
 extracting: test/62754.png
 extracting: test/60523.png
 extracting: test/60305.png
```

```
extracting: test/67378.png
extracting: test/67866.png
extracting: test/63887.png
extracting: test/60439.png
extracting: test/61901.png
extracting: test/67390.png
extracting: test/68877.png
extracting: test/67986.png
extracting: test/65327.png
extracting: test/65421.png
extracting: test/63323.png
extracting: test/61877.png
 inflating: test.csv
```

```
test = pd.read_csv('test.csv')
```

```
test_image = []
for i in tqdm(range(test.shape[0])):
    img = image.load_img('test/'+test['id'][i].astype('str')+'.png', target_size=(28,28,1)
    img = image.img_to_array(img)
    img = img/255
    test_image.append(img)
test = np.array(test_image)
```

```
100%|██████████| 10000/10000 [00:01<00:00, 5052.63it/s]
```

```
# making predictions
predictions = (model.predict(test) > 0.5).astype("int32")
```

```
313/313 [==============================] - 5s 17ms/step
```

```
download = drive.CreateFile({'id': '1s79g2s8hu40CSuPf1O03O7SxQXSOneR9'})
download.GetContentFile('sample_submission_I5njJSF.csv')
```

```
# creating submission file
sample = pd.read_csv('sample_submission_I5njJSF.csv')

sample['label'] = predictions
sample.to_csv('sample_cnn.csv', header=True, index=False)
```

sample_cnn - Notepad

File Edit View

```
id,label
60001,0
60002,0
60003,0
60004,0
60005,0
60006,0
60007,0
60008,0
60009,0
60010,0
60011,0
60012,0
60013,0
60014,0
60015,0
60016,0
60017,0
60018,0
60019,0
60020,1
60021,0
60022,0
60023,0
60024,0
60025,0
60026,0
```

Ln 1, Col 1 | 100% | Unix (LF) | UTF-8

sample_cnn - Notepad

File Edit View

```
69974,0
69975,0
69976,0
69977,0
69978,1
69979,0
69980,0
69981,1
69982,1
69983,0
69984,0
69985,0
69986,0
69987,0
69988,0
69989,0
69990,0
69991,0
69992,0
69993,0
69994,0
69995,0
69996,0
69997,0
69998,0
69999,0
70000,0
```

Ln 1, Col 1 | 100% | Unix (LF) | UTF-8

Assignment No.4

Title : ECG Anomaly detection using Autoencoders

Aim: 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

Theory :

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

Sample Code with comments : Attach Printout with Output .

Conclusion: In such a way we use Autoencoder to implement anomaly

detection. To the build required model.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
import tensorflow as tf
from tensorflow.keras.models import Model
from sklearn.model_selection import train_test_spli
from sklearn.preprocessing import MinMaxScaler, Sta
mpl.rcParams['figure.figsize'] = (10, 5)
mpl.rcParams['axes.grid'] = False
```

```
df = pd.read_csv('/content/ecg.csv',sep='  ', heade
df.shape
df.head()
```

```
/usr/local/lib/python3.7/dist-packages/pandas/
  return func(*args, **kwargs)
```

	0
0	-0.11252183,-2.8272038,-3.7738969,-4.3497511,-...
1	-1.1008778,-3.9968398,-4.2858426,-4.5065789,-4...
2	-0.56708802,-2.5934502,-3.8742297,-4.5840949,-...
3	0.49047253,-1.9144071,-3.6163638,-4.3188235,-4...
4	0.80023202,-0.87425189,-2.3847613,-3.9732924,-...

```
df = df.add_prefix('c')
df['c0'].value_counts()
```

```
-0.11252183,-2.8272038,-3.7738969,-4.3497511,-
                    1
-0.44567411,-1.5122063,-2.0832507,-2.5276991,-

1
-0.48834966,-1.1363547,-1.4442442,-1.8557831,-

1
-0.84150524,-1.4423826,-2.0044937,-2.3727734,-
                         1
-0.59911187,-1.303313,-1.5804909,-2.1349974,-2

1

..
-0.75541389,-3.0677909,-3.7511964,-4.222555,-3
    1
-2.2010716,-3.7567285,-4.3434305,-4.4738208,-3
          1
-0.69091917,-2.3131751,-3.0656901,-4.2292249,-
          1
0.64904008,-1.1281441,-2.7718261,-4.0229118,-3
```

Files

..
sample_data
ecg.csv

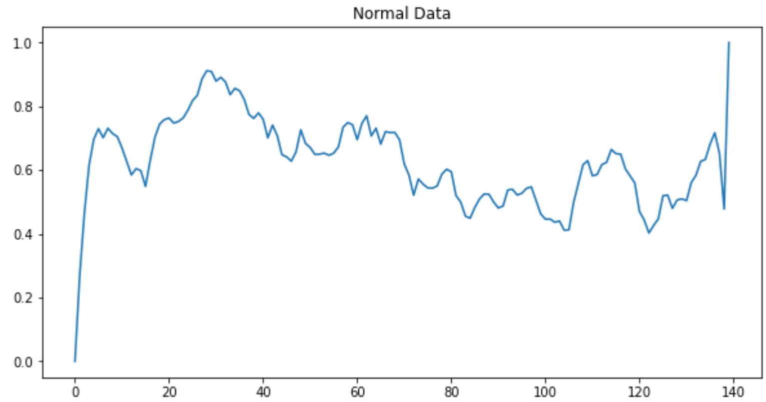
```
                                          1
    -1.3517791,-2.2090058,-2.5202247,-3.0614751,-3
                                                  1
    Name: c0, Length: 4998, dtype: int64
```

```
x_train, x_test, y_train, y_test = train_test_split
```

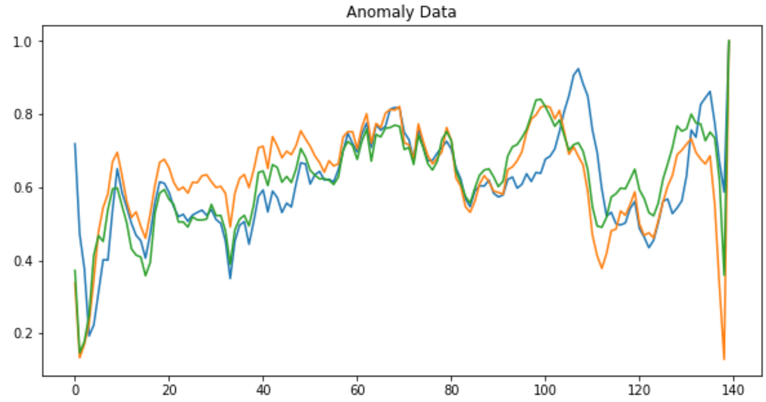
```
scaler = MinMaxScaler()
data_scaled = scaler.fit(x_train)
train_data_scaled = data_scaled.transform(x_train)
test_data_scaled = data_scaled.transform(x_test)
```

```
normal_train_data = pd.DataFrame(train_data_scaled)
anomaly_train_data = pd.DataFrame(train_data_scalec
normal_test_data = pd.DataFrame(test_data_scaled).a
anomaly_test_data = pd.DataFrame(test_data_scaled).
```

```
plt.plot(normal_train_data[0])
#plt.plot(normal_train_data[1])
#plt.plot(normal_train_data[2])
plt.title("Normal Data")
plt.show()
```




```
plt.plot(anomaly_train_data[0])
plt.plot(anomaly_train_data[1])
plt.plot(anomaly_train_data[2])
plt.title("Anomaly Data")
plt.show()
```




```
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(64, activation="rel
model.add(tf.keras.layers.Dense(32, activation="rel
model.add(tf.keras.layers.Dense(16, activation="rel
model.add(tf.keras.layers.Dense(8, activation="relu
model.add(tf.keras.layers.Dense(16, activation="rel
model.add(tf.keras.layers.Dense(32, activation="rel
model.add(tf.keras.layers.Dense(64, activation="rel
model.add(tf.keras.layers.Dense(140, activation="si


class AutoEncoder(Model):
  def __init__(self):
    super(AutoEncoder, self).__init__()
    self.encoder = tf.keras.Sequential([
                  tf.keras.layers.Dense(64, activat
                  tf.keras.layers.Dense(32, activat
                  tf.keras.layers.Dense(16, activat
                  tf.keras.layers.Dense(8, activati
               ])
    self.decoder = tf.keras.Sequential([
                  tf.keras.layers.Dense(16, activat
                  tf.keras.layers.Dense(32, activat
                  tf.keras.layers.Dense(64, activat
                  tf.keras.layers.Dense(140, activa
               ])
  def call(self, x):
    encoded = self.encoder(x)
    decoded = self.decoder(encoded)
    return decoded


model = AutoEncoder()
early_stopping = tf.keras.callbacks.EarlyStopping(m
model.compile(optimizer='adam', loss="mae")
history = model.fit(normal_train_data, normal_trair
                    validation_data=(train_data_sca
                    shuffle=True,
                    callbacks=[early_stopping]
                    )
```

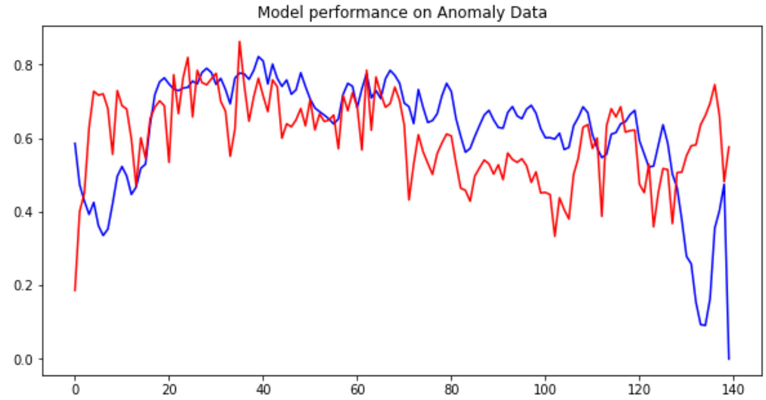
```
Epoch 1/50
1/1 [==============================] - 1s 1s
Epoch 2/50
1/1 [==============================] - 0s 89
Epoch 3/50
1/1 [==============================] - 0s 90
Epoch 4/50
1/1 [==============================] - 0s 89
Epoch 5/50
1/1 [==============================] - 0s 88
Epoch 6/50
1/1 [==============================] - 0s 74
Epoch 7/50
1/1 [==============================] - 0s 79
Epoch 8/50
1/1 [==============================] - 0s 76
Epoch 9/50
1/1 [==============================] - 0s 87
Epoch 10/50
1/1 [==============================] - 0s 73
Epoch 11/50
1/1 [==============================] - 0s 90
Epoch 12/50
1/1 [==============================] - 0s 85
Epoch 13/50
1/1 [==============================] - 0s 72
Epoch 14/50
1/1 [==============================] - 0s 93
Epoch 15/50
1/1 [==============================] - 0s 75
Epoch 16/50
1/1 [==============================] - 0s 85
Epoch 17/50
1/1 [==============================] - 0s 85
Epoch 18/50
1/1 [==============================] - 0s 74
Epoch 19/50
1/1 [==============================] - 0s 72
Epoch 20/50
1/1 [==============================] - 0s 74
Epoch 21/50
1/1 [==============================] - 0s 88
Epoch 22/50
1/1 [==============================] - 0s 87
Epoch 23/50
1/1 [==============================] - 0s 87
Epoch 24/50
1/1 [==============================] - 0s 90
Epoch 25/50
1/1 [==============================] - 0s 90
Epoch 26/50
1/1 [==============================] - 0s 89
Epoch 27/50
1/1 [==============================] - 0s 90
Epoch 28/50
1/1 [==============================] - 0s 70
Epoch 29/50
```

```
encoder_out = model.encoder(normal_test_data).numpy
```

```
decoder_out = model.decoder(encoder_out).numpy()

encoder_out_a = model.encoder(anomaly_test_data).nu
decoder_out_a = model.decoder(encoder_out_a).numpy(


plt.plot(anomaly_test_data[0], 'b')
plt.plot(decoder_out_a[0], 'r')
plt.title("Model performance on Anomaly Data")
plt.show()
```



Disk 69.34 GB available

Colab paid products - Cancel contracts here

0s completed at 12:19

Assignment No.5

Title : Implement the Continuous Bag of Words (CBOW) Model.

Aim: Implement the Continuous Bag of Words (CBOW) Model. Stages can be:
a. Data preparation
b. Generate training data
c. Train model
d. Output

Steps/ 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'])

Sample Code with comments : Attach Printout with Output .

Conclusion: The CBOW model tries to understand the context of the words and takes this as input. It then tries to predict words that are contextually accurate.

```
%matplotlib inline import numpy as np import pandas as pd import sklearn
import matplotlib.pyplot as plt from sklearn.feature_extraction.text import
CountVectorizer from sklearn.metrics import confusion_matrix from
sklearn.model_selection import GridSearchCV from sklearn.model_selection
import train_test_split from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression from
sklearn.naive_bayes import MultinomialNB from sklearn.ensemble import
RandomForestClassifier
```

```
pharses=["The quick brown fox jumped over the lazy dog",
         "education is the key to achieve your goals"]
```

```
vect=CountVectorizer() vect.fit(pharses)
```

```
CountVectorizer()
```

```
print("Vocabulary size:{}".format(len(vect.vocabulary_))) print("Vocabulary content:\n
{}".format(vect.vocabulary_))
```

```
Vocabulary size:15 Vocabulary
content:
                    {'the': 12, 'quick': 11, 'brown': 1, 'fox': 4, 'jumped': 7, 'over': 10, 'lazy': 9,
```

```
bag_of_words=vect.transform(pharses)
```

```
print(bag_of_words)
```

```
(0, 1)      1
(0, 2)      1
(0, 4)      1
(0, 7)      1
(0, 9)      1
(0, 10)     1
(0, 11)     1
(0, 12)     2
(1, 0)      1
(1, 3)      1
(1, 5)      1
(1, 6)      1
(1, 8)      1
(1, 12)     1
(1, 13)     1
(1, 14)     1
```

```
print("bag_of_words as  an  array:\n{}".format(bag_of_words.toarray()))
```

```
bag_of_words as an array:
[[0 1 1 0 1 0 0 1 0 1 1 1 2 0 0]
  [1 0 0 1 0 1 1 0 1 0 0 0 1 1 1]]
```

```
vect.get_feature_names()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarnin warnings.warn(msg,
    category=FutureWarning)
['achieve',
  'brown',
  'dog',
  'education',
  'fox',
  'goals',
  'is',
  'jumped',
  'key', 'lazy', 'over',
  'quick',
  'the',
  'to',
  'your']
```

```
data=pd.read_csv("/content/labeledTrainData.tsv",delimiter="\t")
```

```
data.head()
```

	id	sentiment	review
0	5814_8	1	With all this stuff going down at the moment w...
1	2381_9	1	\The Classic War of the Worlds\" by Timothy Hi...
2	7759_3	0	The film starts with a manager (Nicholas Bell)...
3	3630_4	0	It must be assumed that those who praised this...
4	9495_8	1	Superbly trashy and wondrously unpretentious 8...

```
print("Samples per class;{}".format(np.bincount(data.sentiment)))
```

```
Samples per class;[3048 3144]
```

```
def  simple_split(data,y,length,split_mark=0.7):
   if  split_mark  >0.and  split_mark  <  1.0:
      n=int(split_mark*length)
   else:
      n=int(split_mark)
```

```
    X_train=data[:n].copy() X_test=data[n:].copy()
    y_train=y[:n].copy() y_test=y[n:].copy()
    return X_train,X_test,y_train,y_test
```

```
vectorizer=CountVectorizer()
```

```
X_train,X_test,y_train,y_test = simple_split(data.review,data.sentiment,len(data))
print(X_train.shape,X_test.shape,y_train.shape,y_test.shape)
```

```
(4334,) (1858,) (4334,) (1858,)
```

```
print("Samples per  class:{}".format(np.bincount(y_train))) print("Samples per
class:{}".format(np.bincount(y_test)))
```

```
Samples per class:[2157 2177]
Samples per class:[891 967]
```

```
X_train=vectorizer.fit_transform(X_train)
X_test=vectorizer.transform(X_test)
```

```
feature_names=vectorizer.get_feature_names() print("Number of
features:{}".format(len(feature_names))) print("First 20
features:\n{}".format(feature_names[:20])) print("Features 19500 to
19530:\n{}".format(feature_names[19500:19530])) print("Every 2000th
feature:\n{}".format(feature_names[::2000]))
```

```
Number of features:37042 First 20
features:
['00', '000', '00015', '001', '007', '00pm', '00s', '01', '02', '03', '04', '041', ' Features 19500 to 19530:
['lock', 'locke', 'locked', 'locker', 'lockhart', 'lockheed', 'locking', 'locks', 'l Every 2000th feature:
['00', 'arden', 'bonanza', 'chronicling', 'cunningham', 'drama', 'farr', 'goddess',
```

```
vectorizer.vocabulary_ hammering :
  14829,
 'straight': 31583,
 'through': 33206,
 'earhole': 10392,
 'uses': 34992, 'tired': 33364,
 'comedic': 6609,
 'techniques': 32759,
 'consistently': 7093,
 'breaking': 4322,
 'fourth': 12986,
 'wall': 35693,
 'talks': 32570

 'audience': 2425,
```

'seemingly': 29107,
'pointless': 24988,
'montages': 21621,
'hot': 15910,
'girls': 13834,
'waiter': 35663,
'ship': 29610,
'successful': 31930,
'comedian': 6607,
'order': 23280,
'become': 3135,
'women': 36433,
'resident': 27377,
'shamelessly': 29441,
'named': 22154,
'dickie': 9124,
'due': 10203,
'unfathomable': 34545,
'success': 31928,
'opposite': 23234, 'gender': 13620,
'presumed': 25498,
'lost': 19641,
'sea': 29008, 'shecker': 29512,
'break': 4313,
'rather': 26549,
'locked': 19502,
'bathroom': 2998,
'presumably': 25496, 'perhaps': 24310,
'vomited': 35573,
'worst': 36542,
'references': 26909,
'mad': 19964, 'max':
20597,
'ii': 16294,
'wild': 36214,
'ladybug': 18676,
'clear': 6214,
'reference': 26907,
'tribute': 33899, 'peter': 24444,
'lorre': 19629,
'masterpiece': 20498,

```
i=45000 j=10 words=vectorizer.get_feature_names()[i:+10]
pd.DataFrame(X_train[j:j+7,i:i+10].todense(),columns=words)
```

/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarnin warnings.warn(msg, category=FutureWarning)

0

1

2

3

4 scores=cross_val_score(LogisticRegre
ssion(),X_train,y_train,cv=5) print("Mean cross-
validation accuracy:{:2f}".format(np.mean(scores)))5

/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Conver 6 STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in: https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options: https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Conver STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in: https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options: https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Conver STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in: https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options: https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Conver STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in: https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options: https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
Mean cross-validation accuracy:0.845866
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Conver STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in: https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options: https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,

logreg=LogisticRegression()
logreg.fit(X_train,y_train) print("training set score:{:.3f}".format
(logreg.score(X_train,y_train))) print("Test set
score:{:.3f}".format(logreg.score(X_test,y_test)))

```
training set score:1.000 Test set
score:0.872
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818:     Conver STOP: TOTAL
NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in: https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options: https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
```

```
pred_logreg=logreg.predict(X_test)
confusion=confusion_matrix(y_test,pred_logreg) print("confusin
matrix:\n{}".format(confusion))
```

```
confusin matrix:
[[774 117]
 [120 847]]
```

```
nb=MultinomialNB() nb.fit(X_train,y_train) print("Training set
score:{:.3f}".format(nb.score(X_train,y_train))) print("Test set
score:{:.3f}".format(nb.score(X_test,y_test)))
```

```
Training set score:0.950
Test set score:0.844
```

```
pred_nb=nb.predict(X_test) confusion=confusion_matrix(y_test,pred_nb)
print("Confusion matrix:\n{}".format(confusion))
```

```
Confusion matrix:
[[790 101]
 [189 778]]
```

```
rf=RandomForestClassifier() rf.fit(X_train,y_train)
print("Training set score:{:.3f}".format(rf.score(X_train,y_train))) print("Test set
score:{:.3f}".format(rf.score(X_test,y_test)))
```

```
Training set score:1.000
Test set score:0.847
```

```
review="This movie is not that good" print(logreg.predict(vectorizer.transform([review]))[0])
print(rf.predict(vectorizer.transform([review]))[0]) print(nb.predict(vectorizer.transform([review]))[0])
```

```
0
1
0
```

```
review="This movie is not that bad"
print(logreg.predict(vectorizer.transform([review]))[0]) print(rf.predict(vectorizer.transform([review]))[0])
print(nb.predict(vectorizer.transform([review]))[0])
```

0
0
0

review="i was going to say something awesome or great or good,but i can't"
print(logreg.predict(vectorizer.transform([review]))[0]) print(rf.predict(vectorizer.transform([review]))[0])
print(nb.predict(vectorizer.transform([review]))[0])

1
1
1

param_grid={'C':[0.001,0.001,1,10]}
grid=GridSearchCV(LogisticRegression(),param_grid,cv=5) grid.fit(X_train,y_train)
print("Best cross validation score:{:.2f}".format(grid.best_score_)) print("Best parameters:",grid.best_params_)

extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Conv STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in: https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options: https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Conv STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in: https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options: https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Conv STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in: https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options: https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Conv STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in: https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options: https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Conv STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in: https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options: https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Conv STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in: https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options: https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
Best cross validation score:0.85
Best parameters: {'C': 1}
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Conv STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in: https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

Assignment No.6

Title : Object detection using Transfer Learning of CNN architectures

Aim: Object detection using Transfer Learning of CNN architectures
a. Load in a pre-trained CNN model trained on a large dataset
b. Freeze parameters (weights) in model's lower convolutional layers
c. Add custom classifier with several layers of trainable parameters to model
d. Train classifier layers on training data available for task
e. Fine-tune hyper parameters and unfreeze more layers as needed

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)
```

Sample Code with comments : Attach Printout with Output .

Conclusion: The main benefits of transfer learning include the saving of resources and improved efficiency when training new models. It can also help with training models when only unlabelled datasets are available, as the bulk of the model will be pre-trained.

https://www.google.com/url?q=https://towardsdatascience.com/transfer-learning-with-convolutional-neural-networks-in-pytorch-dd0

In [1]:

```
from IPython.core.interactiveshell import InteractiveShell
import seaborn as sns
# PyTorch
from torchvision import transforms, datasets, models
import torch
from torch import optim, cuda
from torch.utils.data import DataLoader, sampler
import torch.nn as nn

import warnings
warnings.filterwarnings('ignore', category=FutureWarning)

# Data science tools
import numpy as np
import pandas as pd
import os

# Image manipulations
from PIL import Image
# Useful for examining network
from torchsummary import summary
# Timing utility
from timeit import default_timer as timer

# Visualizations
import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams['font.size'] = 14

# Printing out all outputs
InteractiveShell.ast_node_interactivity = 'all'
```

In [2]:

```python
# Location of data
datadir = '/home/wjk68/'
traindir = datadir + 'train/'
validdir = datadir + 'valid/'
testdir = datadir + 'test/'

save_file_name = 'vgg16-transfer-4.pt'
checkpoint_path = 'vgg16-transfer-4.pth'

# Change to fit hardware
batch_size = 128

# Whether to train on a gpu
train_on_gpu = cuda.is_available()
print(f'Train on gpu: {train_on_gpu}')

# Number of gpus
if train_on_gpu:
    gpu_count = cuda.device_count()
    print(f'{gpu_count} gpus detected.')
    if gpu_count > 1:
        multi_gpu = True
    else:
        multi_gpu = False
```

```
Train on gpu: True
2 gpus detected.
```

In [3]:

```
# Empty lists
categories = []
img_categories = []
n_train = []
n_valid = []
n_test = []
hs = []
ws = []

# Iterate through each category
for d in os.listdir(traindir):
    categories.append(d)

    # Number of each image
    train_imgs = os.listdir(traindir + d)
    valid_imgs = os.listdir(validdir + d)
    test_imgs = os.listdir(testdir + d)
    n_train.append(len(train_imgs))
    n_valid.append(len(valid_imgs))
    n_test.append(len(test_imgs))

    # Find stats for train images
    for i in train_imgs:
        img_categories.append(d)
        img = Image.open(traindir + d + '/' + i)
        img_array = np.array(img)
        # Shape
        hs.append(img_array.shape[0])
        ws.append(img_array.shape[1])

# Dataframe of categories
cat_df = pd.DataFrame({'category': categories,
                       'n_train': n_train,
                       'n_valid': n_valid, 'n_test': n_test}).\
    sort_values('category')

# Dataframe of training images
image_df = pd.DataFrame({
    'category': img_categories,
    'height': hs,
    'width': ws
})

cat_df.sort_values('n_train', ascending=False, inplace=True)
```

```
cat_df.head()
cat_df.tail()
```

Out[3]:

	category	n_train	n_valid	n_test
4	airplanes	400	200	200
2	motorbikes	398	200	200
0	faces	274	138	109
93	watch	119	60	60
1	leopards	100	50	50

Out[3]:

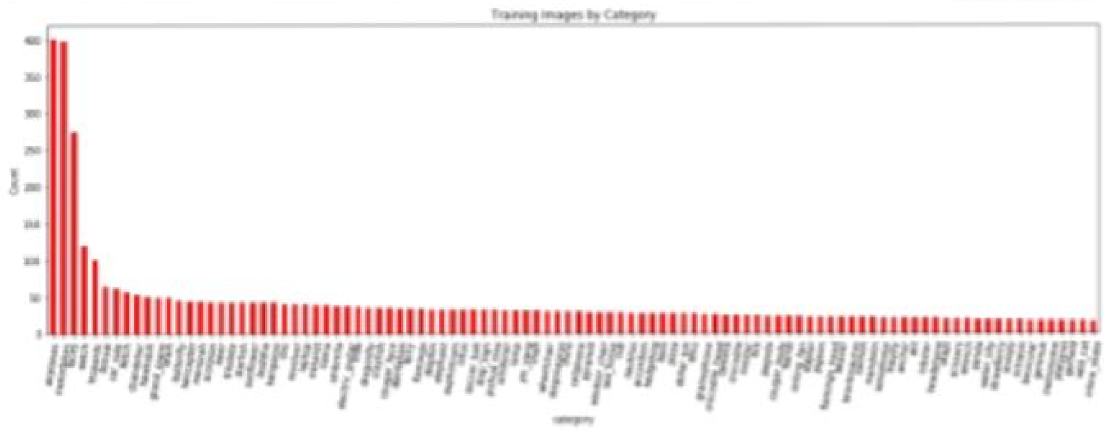
	category	n_train	n_valid	n_test
63	metronome	16	8	8
72	platypus	16	9	9
42	garfield	16	9	9
96	wild_cat	16	9	9
51	inline_skate	15	8	8

Distribution of Images

There are between 400 and 15 training images in each category. The low number of training images may result in reduced scores in some categories.

In [4]:

```
cat_df.set_index('category')['n_train'].plot.bar(
    color='r', figsize=(20, 6))
plt.xticks(rotation=80)
plt.ylabel('Count')
plt.title('Training Images by Category')
```



In [5]:

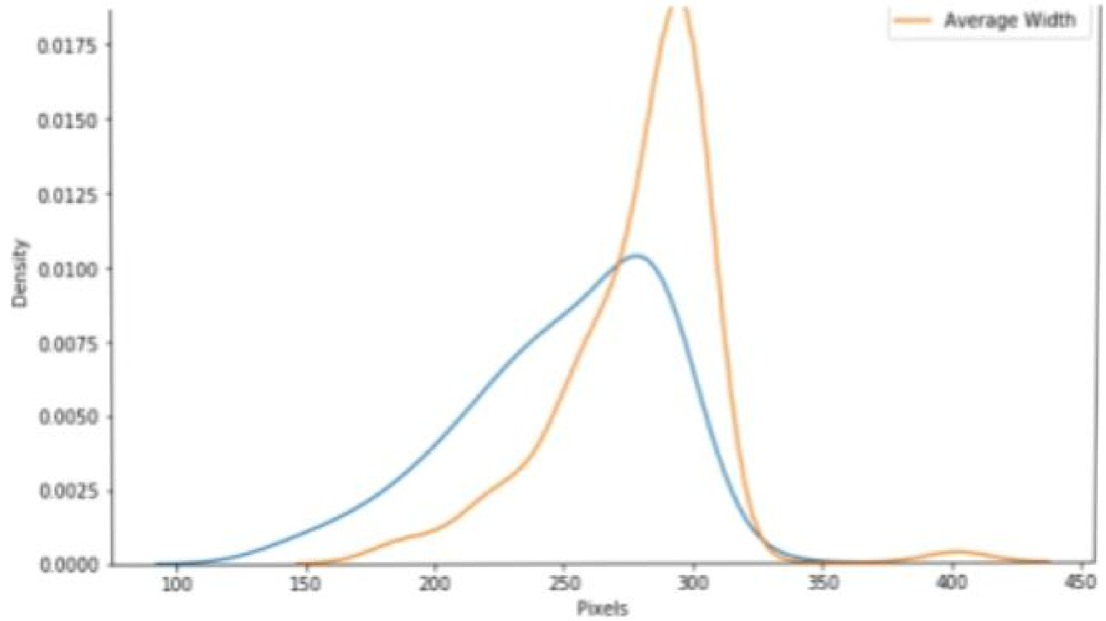
```
# Only top 50 categories
cat_df.set_index('category').iloc[:50]['n_train'].plot.bar(
    color='r', figsize=(20, 6))
plt.xticks(rotation=80)
plt.ylabel('Count')
```

In [6]:
```python
img_dsc = image_df.groupby('category').describe()
img_dsc.head()
```

Out[6]:

	height									
	count	mean	std	min	25%	50%	75%	max	count	m
category										
accordion	27.0	263.851852	35.769243	199.0	233.00	265.0	300.00	300.0	27.0	280.333
airplanes	400.0	158.455000	30.847397	101.0	141.00	154.0	170.25	494.0	400.0	402.137
anchor	20.0	241.000000	38.608698	170.0	219.75	236.0	264.50	300.0	20.0	291.300
ant	20.0	211.950000	47.137509	103.0	177.00	203.0	236.75	300.0	20.0	298.600
barrel	23.0	284.086957	36.455344	188.0	300.00	300.0	300.00	300.0	23.0	241.869

In [7]:
```python
plt.figure(figsize=(10, 6))
sns.kdeplot(
    img_dsc['height']['mean'], label='Average Height')
sns.kdeplot(
    img_dsc['width']['mean'], label='Average Width')
plt.xlabel('Pixels')
plt.ylabel('Density')
plt.title('Average Size Distribution')
```



When we use the images in the pre-trained network, we'll have to reshape them to 224 x 224. This is the size of Imagenet images and is therefore what the model expects. The images that are larger than this will be truncated while the smaller images will be interpolated.

In [8]:

```
def imshow(image):
    """Display image"""
    plt.figure(figsize=(6, 6))
    plt.imshow(image)
    plt.axis('off')
    plt.show()


# Example image
x = Image.open(traindir + 'ewer/image_0002.jpg')
np.array(x).shape
imshow(x)
```

Out[8]: (300, 187, 3)

values are standardized for Imagenet.

In [10]:

```
# Image transformations
image_transforms = {
    # Train uses data augmentation
    'train':
    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 does not use augmentation
    'val':
    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])
    ]),
    # Test does not use augmentation
    'test':
    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])
    ]),
}
```

```
In [11]:  def imshow_tensor(image, ax=None, title=None):
              """Imshow for Tensor."""

              if ax is None:
                  fig, ax = plt.subplots()

              # Set the color channel as the third dimension
              image = image.numpy().transpose((1, 2, 0))

              # Reverse the preprocessing steps
              mean = np.array([0.485, 0.456, 0.406])
              std = np.array([0.229, 0.224, 0.225])
              image = std * image + mean

              # Clip the image pixel values
              image = np.clip(image, 0, 1)
```



```
              ax.imshow(image)
              plt.axis('off')

              return ax, image
```

We'll work with two example images and apply the train transformations.

```
In [12]:  ex_img = Image.open('/home/wjk68/train/elephant/image_0024.jpg')
          imshow(ex_img)
```

```
In [13]:  t = image_transforms['train']
          plt.figure(figsize=(24, 24))

          for i in range(16):
              ax = plt.subplot(4, 4, i + 1)
              _ = imshow_tensor(t(ex_img), ax=ax)

          plt.tight_layout()

Out[13]:  <Figure size 1728x1728 with 0 Axes>
```