Index

Sr No.	Title of Experiment	Date	Marks	Signature
1	Study of Deep learning Packages.	08-Aug-22	10	
2	Implementing Feedforward neural networks with Keras and TensorFlow	22-Aug-22	10	
3	Build the Image classification model by dividing the model	29-Aug-22	10	
4	Use Autoencoder to implement anomaly detection.	12-Sep-22	10	
5	Implement the Continuous Bag of Words (CBOW) Model.	19-Sep-22	10	
6	Object detection using Transfer Learning of CNN architectures.	03-Oct-22	10	

Page No. 4
Date

AIM: Study of deep learning packaging : Tensorflow, Keras, Theano and PyTorch. Document the distinct features and functionality of the packages.

OBJECTIVE : Introduction to various deep learning tools and how to use them.

Software used : Windows/Linux/Mac, Tensorflow, Keras, PyTorch

THEORY

Tensorflow

- Tensorflow is an open source software library released in 2015 by Google to make it easier for developers to design, build, and train deep learning models. Tensorflow originated as an internal library that Google developers used to build models in-house, and we expect additional functionality to be added to the open source version as it is tested and vetted in the internal flavour.

On high level, Tensorflow is a Python library that allows users to express arbitrary computations as a graphs of data flows. Nodes in the graphs represent mathematical operations, whereas edges represent data that is communicated from one node to another. Data in TensorFlow is represented as tensors, which are ~~multip~~ multidimensional arrays.

Running Tensorflow

```
import tensorflow as tf
deep_learning = tf.constant('Deeplearning')
session = tf.Session()
```

```
session.run(deep_learning)
a = tf.constant(2)
a = tf.constant(2)
multiply = tf.mul(a,b)
session.run(multiply)
```

Tensorflow Operations

Category	Examples
Element-wise mathematical op	Add, Sub, Mul, Div, Exp, Log, Greater, Less, ...
Array operations	Concat, Slice, Split, Constant, Rank, Shape, ...
Matrix operations	MatMul, MatrixInverse, MatrixDeterminant
Stateful operations	Variable, Assign, AssignAdd, ...
Neural network building blocks	SoftMax, Sigmoid, ReLU, Convolution2D, Max...
Checkpointing operations	Save, Restore
Queue and synchronization op	Enqueue, Dequeue, MutexAcquire, MutexRelease
Control flow operations	Merge, Switch, Enter, Leave, NextIteration

Keras

Keras is an open source deep learning framework for python. It has been developed by an AI researcher at Google named Francois Chollet.

Keras is based on minimal structure that provides a clean and easy way to create deep learning models based on Tensorflow or Theano. Keras is designed to quickly define deep learning models. Well

Features

Keras leverages various optimization techniques to make high level neural network API easier, and more performant. It supports

following features -

- Consistent, simple, and extensible API.
- Minimal structure - easy to achieve the result without any frills
- It supports multiple platforms and backends.
- It is user friendly framework which runs on both CPU and GPU.
- Highly scalability of computation.

Benefits:

Keras is highly powerful and dynamic framework and comes up with the following advantages-

- Larger community support
- Easy to test.
- Keras neural networks are written in Python which makes things simple
- Keras support of both convolution and recurrent networks.
- Deep learning models are discrete components, so that you can combine many ways.

PyTorch:

PyTorch is an optimized tensor library for deep learning using GPUs and CPUs. PyTorch is an optimized DeepLearning tensor library based on Python, and Torch is mainly used for applications using GPUs and CPUs. PyTorch is favored over other DeepLearning frameworks like TensorFlow and Keras since it uses dynamic computation graph and is completely Pythonic.

Two main features of PyTorch are:

- Tensor computation with strong GPU acceleration support
- Automatic Differentiation for creating and training deep neural networks.

Page No. 7
Date

Common PyTorch Modules

In pyTorch, modules are used to represent neural networks.

i. Autograd

- The autograd module is PyTorch's automatic differentiation that helps to compute the gradients in the forward pass quick time. Autograd generates a directed acyclic graph (D leaves are the input tensors while the roots are the outpu

ii. Optim

- The optim module is a package with pre-written algo for optimizers that can be used to build neural networks.

iii. nn

- The nn module includes various classes that help to bui neural network models. All modules in PyTorch subclass nn module.

Conclusion: We learnt about the various deep learning tra

Page No. 8
Date

ASSIGNMENT NO. 2

Name: Akhil. A
Roll No.: 004
Subject: LP-IV (DL)

BE-IT 2022-23
DOP: 08 Aug 2022 DOC: 22 Aug 2022
Grade: 10/10 Sign:

Page No. 9
Date

Aim : Implementing ~~Feedforward~~ feedforward neural networks with Keras and Tensorflow.

a. Import the necessary packages.
b. Load the training and testing data
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 : To learn how to develop a feedforward neural network and how to optimize it for better performance

Infrastructure : Computer / Laptop / Virtual Machine

Software used : Jupyter Notebook / Google colab / Tensorflow and Keras

Theory :
What is Feed forward neural networks ?
Deep feedforward networks, also often called feedforward neural networks, or multilayer perceptrons. The goal of a feedforward network is to approximate some function f^*. For example, for a classifier, $y = f^*(x)$ maps an input x to a category y. A feedforward network defines a mapping $y = f(x, \theta)$ and learns the value of the parameters θ that result in the best function approximation.

These models are called feedforward because information flows

through the function being evaluated from x, through the intermediate computations used to define f, and finally to the output y. There are no feedback connections in which outputs of the models are fed back into itself.

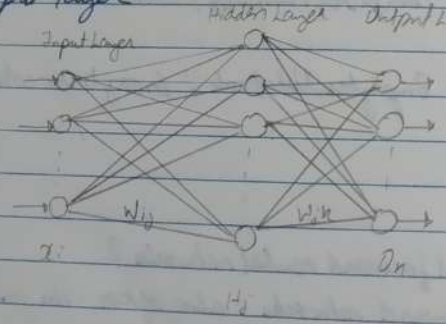
Structure of feedforward Neural network

The basic structure consists of the following layers:

i. Input layer: It is where the user accepts the input for the neural network.

ii. Hidden layer: This is the layer where all the computations required for the prediction are done.

iii. Output layer: The output from the hidden layer is provided at the output layer.



The nodes are connected with the help of edges. The edge is represented by W_{ij} where i represent the node where the edge starts from and j represents the node where the edge ends.

The nodes compute one output for the next layer by taking in the product of the node input and the weight associated with the node edge, which is then applied to an activation function

decide whether the node should fire or not for the output.

SGD

Stochastic gradient descent (SGD) and its variants are probably the most used optimization algorithms for ML in general and for deep learning in particular. A crucial parameter for the SGD algorithm is the learning rate.

The standard gradient descent algorithm updates the parameters θ. θ of the objective $J(\theta)J(\theta)$ as

$$\theta = \theta = \alpha \nabla\theta E[J(\theta)] \theta = \theta - \alpha \nabla\theta[J(\theta)]$$

where the expectation in the above equation is approximated by evaluating the cost and gradient over the full training set.

MNIST / CIFAR 10

MNIST: The MNIST dataset of handwritten digits has training set of 70,000 examples and each row of the matrix corresponding to a 28x28 images. The unique values of the response variable y range from 0 to 9.

CIFAR 10: CIFAR-10 is an extra established computer vision dataset used for object recognition. The dataset have 80 million tiny images dataset and consist of 60,000; 32x32 colour images containing one of 10 object classes. Furthermore, the data were converted from RGB to gray, normalized and rounded to 2 decimal places.

Page No. 12
Date

Implementation

i. Load the necessary libraries
ii. Load the datasets from the libaries or from outside.
iii. Build the feedforward neural network using Keras.
iv. Train the model with the dataset and use SGD as optimizer
v. Evaluate the model for the accuracy and other evaluation met
vi. Plot the loss and accuracy function.

Conclusion: We developed a feedforward neural network for handwritten digit recognition.

Name: Akhil A

Roll No.: 004

Subject: LP-IV(DL)

In [1]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout, Flatten
import matplotlib.pyplot as plt
# import seaborn as sns
```

MNIST dataset

In [2]:
```
mnist = tf.keras.datasets.mnist
(x_train, y_train) , (x_test, y_test) = mnist.load_data() # Data loading
x_train, x_test = x_train/255.0 , x_test/255.0 #Normalizing the data
```

In [27]:
```
sns.heatmap(x_train[0])
plt.show()
```

Prepearing the model

In [3]:
```
model = Sequential([
    Flatten(input_shape=(28,28)),
    Dense(128, activation="relu"),
    Dropout(0.2),
    Dense(10)
])
```

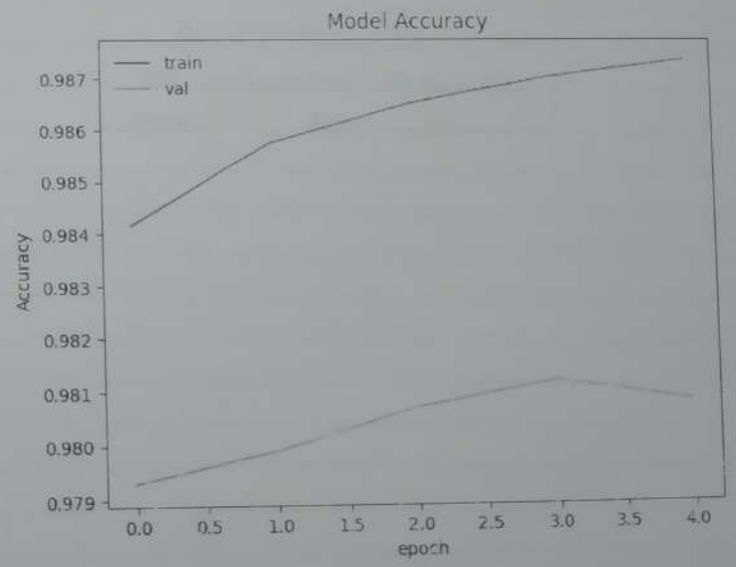
```
In [5]: predictions = model(x_train[:1]).numpy()
        predictions

Out[5]: array([[-0.24707137, -0.64293617,  0.32793105, -0.7325163 , -0.10029303,
                 0.42578584, -0.5628654 , -0.9137927 , -1.2458755 ,  0.75219357]],
              dtype=float32)

In [6]: tf.nn.softmax(predictions).numpy()

Out[6]: array([[0.08687506, 0.0584754 , 0.15438868, 0.05346493, 0.10060976,
                0.17026025, 0.06335013, 0.0446007 , 0.03199779, 0.23597735]],
              dtype=float32)

In [8]: loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)

In [9]: model.compile(optimizer="adam", loss = loss_fn, metrics=["accuracy"])

In [10]: model.fit(x_train, y_train, epochs=5)

         Epoch 1/5
         1875/1875 [==============================] - 5s 3ms/step - loss: 0.2930 - accur
         acy: 0.9143
         Epoch 2/5
         1875/1875 [==============================] - 5s 3ms/step - loss: 0.1412 - accur
         acy: 0.9575
         Epoch 3/5
         1875/1875 [==============================] - 5s 3ms/step - loss: 0.1048 - accur
         acy: 0.9683
         Epoch 4/5
         1875/1875 [==============================] - 5s 3ms/step - loss: 0.0855 - accur
         acy: 0.9736
         Epoch 5/5
         1875/1875 [==============================] - 5s 3ms/step - loss: 0.0729 - accur
         acy: 0.9769

Out[10]: <keras.callbacks.History at 0x27c0bc71210>

In [11]: model.evaluate(x_test, y_test, verbose=2)

         313/313 - 1s - loss: 0.0750 - accuracy: 0.9764 - 849ms/epoch - 3ms/step

Out[11]: [0.07503402578181839, 0.9764000177383423]
```

Validation of Model

In [12]:
```
val = model.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_test), bat
```

```
Epoch 1/5
300/300 [==============================] - 1s 4ms/step - loss: 0.0516 - accurac
y: 0.9841 - val_loss: 0.0659 - val_accuracy: 0.9793
Epoch 2/5
300/300 [==============================] - 1s 4ms/step - loss: 0.0456 - accurac
y: 0.9858 - val_loss: 0.0641 - val_accuracy: 0.9799
Epoch 3/5
300/300 [==============================] - 1s 4ms/step - loss: 0.0437 - accurac
y: 0.9865 - val_loss: 0.0649 - val_accuracy: 0.9807
Epoch 4/5
300/300 [==============================] - 1s 4ms/step - loss: 0.0415 - accurac
y: 0.9870 - val_loss: 0.0633 - val_accuracy: 0.9812
Epoch 5/5
300/300 [==============================] - 1s 4ms/step - loss: 0.0398 - accurac
y: 0.9874 - val_loss: 0.0627 - val_accuracy: 0.9808
```

In [13]:
```
plt.title("Model Accuracy")
plt.ylabel("Accuracy")
plt.xlabel("epoch")
plt.plot(val.history["accuracy"])
plt.plot(val.history["val_accuracy"])
plt.legend(["train","val"])
plt.show()
```



In [ ]:

Page No. 16
Date

ASSIGNMENT NO. 3

Name : Akhil. A
Roll NO. : 004
Subject : LP-IV (OL)

BE - IT 2022 - 23
DOP : 22 Aug 2022 DOC : 29 Aug 2022
Marks : 10/10 Sign :

AIM: Build the image classification model by dividing the model into the following 4 stages:
a. Loading the and preprocessing the image data.
b. Defining the model's architecture
c. Training the model
d. Estimating the model's performance.

Objective: To learn about CNN and how to develop a CNN for image recognition.

Infrastructure: Computer / Laptop / Virtual Machine

Software used: Jupyter Notebook / Google Colab, Tensorflow, Keras

THEORY

CNN

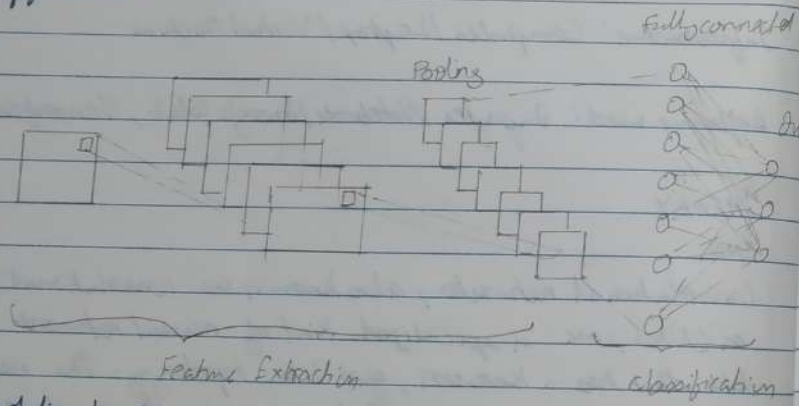
Convolutional networks, also known as convolutional neural network or CNNs, are a specialized kind of neural network for processing data that has a known, grid-like topology. The name "convolutional neural network" indicates that the network employs a mathematical operation called convo convolution. Convolution is a specialized kind of linear operation. Convolution networks are simply neural networks that use convolution in place of general matrix multiplication in atleast one of their layers.

Architecture of CNN

- There are two main parts of a CNN a:
- A convolution tool that seperates and identifies the various features

of the image for analysis in a process called as Feature E

- The network of feature extraction consists of many pairs convolutional or pooling layers.
- A fully connected layer that utilizes the output from the ~~a pooling layer~~ process and predicts the class of the image on the features extracted in previous stages.
- This CNN models of feature extraction aims to reduce the of features present in a dataset. It creates new features summarises the existing features contained in an origin of features.



Convolution Layer

There are three types of layers that make up the CNN which are convolutional layers, pooling layers, and fully-connect (FC) la In addition to these layers there are two more imp important parameter which are the dropout layer and the a function.

1. Convolutional Layer

This layer is the first layer that is used to extract

features from the input images. In this layer, the mathematical operation of convolution is performed between the input image and a filter of a particular size MxM. By sliding the filter over the input image, the dot product is taken between the filter and the parts of the input image with respect to the size of the filter (MxM).

- The output is termed as the feature map which gives us information about the image such as the corners and edges.

ii. Pooling Layer

The primary aim of this layer is to decrease the size of the convolved feature map to reduce the computational costs. This is performed by decreasing the connections between layers and independently operates on each feature map. In Max Pooling, the largest element is taken from feature map. Average pooling calculates the average of the elements in a predefined sized image section. The total sum of the elements in the predefined section is computed in sum pooling.

iii. Fully Connected Layer

- The fully connected (FC) layer consists of the weights and biases along with the neurons and is used to connect the neurons between two different layers. These layers are usually placed before the output layer and form the last few layers of a CNN Architecture.

iv. Dropout

Usually, when all the features are connected to the FC layer, it can cause overfitting in the training dataset. Overfitting occurs when a particula

model works so on the training data causing a negative imp
the models performance when used on a new data.

v. Activation function
They are used to learn and approximate any kind of conti
and complex relationship between variables of the network
In simple words, it decides which information of the mo
should fire in the forward direction and which ones should
not at the end of the network.

Implementation
i. Load the necessary libraries.
ii. Import the dataset from the respective library or local da
iii. Design the neural network architecture and mention the
of layers, nodes, etc.
iv. Then train the model with the imported dataset.
v. Evaluate the performance of the model

Conclusion :
We learnt how to build and train a CNN to identify imag

Name:- Akhil A
Roll No:- 04
PRN No:- 7203613SE
Class:- BE[IT]
Subject:- Deep Learning

In [1]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout
from tensorflow.keras.optimizers import SGD
import numpy as np
import random
import matplotlib.pyplot as plt
import tensorflow as tf
```

In [2]:
```
import os
os.environ['CUDA_VISIBLE_DEVICES'] = '-1'
```

Load Dataset

Dataset avalaible on https://bit.ly/ImgClsKeras (https://bit.ly/ImgClsKeras)

In [3]:
```
x_train = np.loadtxt('input.csv',delimiter = ',')
y_train = np.loadtxt('labels.csv',delimiter = ',')

x_test = np.loadtxt('input_test.csv',delimiter = ',')
y_test = np.loadtxt('labels_test.csv',delimiter = ',')
```

In [4]:
```
x_train = x_train.reshape(len(x_train),100,100,3)
y_train = y_train.reshape(len(y_train),1)

x_test = x_test.reshape(len(x_test),100,100,3)
y_test = y_test.reshape(len(y_test),1)

x_train = x_train/255.0
x_test = x_test/255.0
```

In [5]:
```
print("Shape of X_train:", x_train.shape)
print("Shape of Y_train:", y_train.shape)
print("Shape of X_train:", x_test.shape)
print("Shape of X_train:", y_test.shape)
```

```
Shape of X_train: (2000, 100, 100, 3)
Shape of Y_train: (2000, 1)
Shape of X_train: (400, 100, 100, 3)
Shape of X_train: (400, 1)
```

In [6]:
```
idx = random.randint(0,len(x_train))
plt.imshow(x_train[idx,:])
plt.show()
```

Model Building

In [8]:
```
model = Sequential([
    Conv2D(256,(3,3),activation = 'relu',input_shape=(100,100,3)),
    BatchNormalization(),
    MaxPooling2D((4,4)),
    Conv2D(128,(3,3),activation = 'relu'),
    BatchNormalization(),
    MaxPooling2D((2,2)),
    Conv2D(64,(3,3),activation = 'relu'),
    BatchNormalization(),
    MaxPooling2D((2,2)),

    Flatten(),
    Dense(128,activation='relu'),
    Dropout(0.4),
    Dense(1,activation='sigmoid')
])
```

In [9]:
```
opt = SGD(momentum=0.9)
model.compile(optimizer=opt, loss='binary_crossentropy', metrics = ['accuracy'])
```

```
In [10]: model.fit(x_train,y_train,epochs=30,steps_per_epoch = 20,validation_data=(x_test,

         Epoch 1/30
         20/20 [==============================] - 61s 3s/step - loss: 0.8490 - accuracy:
         0.5650 - val_loss: 0.6948 - val_accuracy: 0.5250
         Epoch 2/30
         20/20 [==============================] - 70s 4s/step - loss: 0.6606 - accuracy:
         0.6240 - val_loss: 0.6801 - val_accuracy: 0.5975
         Epoch 3/30
         20/20 [==============================] - 68s 3s/step - loss: 0.6057 - accuracy:
         0.6830 - val_loss: 0.6797 - val_accuracy: 0.5625
         Epoch 4/30
         20/20 [==============================] - 64s 3s/step - loss: 0.5674 - accuracy:
         0.6990 - val_loss: 0.7254 - val_accuracy: 0.5050
         Epoch 5/30
         20/20 [==============================] - 64s 3s/step - loss: 0.5089 - accuracy:
         0.7605 - val_loss: 0.7999 - val_accuracy: 0.5050
         Epoch 6/30
         20/20 [==============================] - 62s 3s/step - loss: 0.4879 - accuracy:
         0.7720 - val_loss: 0.8164 - val_accuracy: 0.4950
         Epoch 7/30
         20/20 [==============================] - 61s 3s/step - loss: 0.4430 - accuracy:
         0.7960 - val_loss: 0.8590 - val_accuracy: 0.5125
         Epoch 8/30
         20/20 [==============================] - 60s 3s/step - loss: 0.4141 - accuracy:
         0.8130 - val_loss: 0.8978 - val_accuracy: 0.5050
         Epoch 9/30
         20/20 [==============================] - 65s 3s/step - loss: 0.3859 - accuracy:
         0.8270 - val_loss: 1.0060 - val_accuracy: 0.5025
         Epoch 10/30
         20/20 [==============================] - 59s 3s/step - loss: 0.3405 - accuracy:
         0.8470 - val_loss: 1.0907 - val_accuracy: 0.5200
         Epoch 11/30
         20/20 [==============================] - 59s 3s/step - loss: 0.3248 - accuracy:
         0.8555 - val_loss: 0.9742 - val_accuracy: 0.5300
         Epoch 12/30
         20/20 [==============================] - 59s 3s/step - loss: 0.2908 - accuracy:
         0.8820 - val_loss: 0.8911 - val_accuracy: 0.5650
         Epoch 13/30
         20/20 [==============================] - 59s 3s/step - loss: 0.2615 - accuracy:
         0.8920 - val_loss: 0.6965 - val_accuracy: 0.6225
         Epoch 14/30
         20/20 [==============================] - 59s 3s/step - loss: 0.2234 - accuracy:
         0.9145 - val_loss: 0.7834 - val_accuracy: 0.6000
         Epoch 15/30
         20/20 [==============================] - 60s 3s/step - loss: 0.1961 - accuracy:
         0.9245 - val_loss: 1.3311 - val_accuracy: 0.5475
         Epoch 16/30
         20/20 [==============================] - 60s 3s/step - loss: 0.1657 - accuracy:
         0.9470 - val_loss: 1.0894 - val_accuracy: 0.5900
         Epoch 17/30
         20/20 [==============================] - 60s 3s/step - loss: 0.1325 - accuracy:
         0.9590 - val_loss: 1.3612 - val_accuracy: 0.5525
         Epoch 18/30
         20/20 [==============================] - 60s 3s/step - loss: 0.1255 - accuracy:
         0.9565 - val_loss: 1.0850 - val_accuracy: 0.6050
         Epoch 19/30
```

```
20/20 [==============================] - 60s 3s/step - loss: 0.1015 - accuracy:
0.9620 - val_loss: 1.1502 - val_accuracy: 0.5850
Epoch 20/30
20/20 [==============================] - 63s 3s/step - loss: 0.0923 - accuracy:
0.9660 - val_loss: 1.0095 - val_accuracy: 0.6275
Epoch 21/30
20/20 [==============================] - 61s 3s/step - loss: 0.0725 - accuracy:
0.9770 - val_loss: 1.1029 - val_accuracy: 0.6150
Epoch 22/30
20/20 [==============================] - 60s 3s/step - loss: 0.0776 - accuracy:
0.9725 - val_loss: 0.7184 - val_accuracy: 0.7275
Epoch 23/30
20/20 [==============================] - 61s 3s/step - loss: 0.1011 - accuracy:
0.9610 - val_loss: 0.9714 - val_accuracy: 0.6525
Epoch 24/30
20/20 [==============================] - 61s 3s/step - loss: 0.0831 - accuracy:
0.9705 - val_loss: 0.9042 - val_accuracy: 0.6775
Epoch 25/30
20/20 [==============================] - 61s 3s/step - loss: 0.0670 - accuracy:
0.9760 - val_loss: 1.0374 - val_accuracy: 0.6825
Epoch 26/30
20/20 [==============================] - 61s 3s/step - loss: 0.0483 - accuracy:
0.9835 - val_loss: 0.8355 - val_accuracy: 0.7250
Epoch 27/30
20/20 [==============================] - 61s 3s/step - loss: 0.0448 - accuracy:
0.9860 - val_loss: 1.0373 - val_accuracy: 0.7000
Epoch 28/30
20/20 [==============================] - 61s 3s/step - loss: 0.0401 - accuracy:
0.9870 - val_loss: 0.8931 - val_accuracy: 0.7225
Epoch 29/30
20/20 [==============================] - 61s 3s/step - loss: 0.0376 - accuracy:
0.9900 - val_loss: 1.1096 - val_accuracy: 0.7050
Epoch 30/30
20/20 [==============================] - 61s 3s/step - loss: 0.0272 - accuracy:
0.9940 - val_loss: 1.0204 - val_accuracy: 0.7100
```

Out[10]: <keras.callbacks.History at 0x1c39aeacee0>

In [11]:
```
model.evaluate(x_test,y_test)
```

```
13/13 [==============================] - 2s 164ms/step - loss: 1.0204 - accurac
y: 0.7100
```

Out[11]: [1.0204460620880127, 0.7099999785423279]

Making Predictions

In [14]:
```
idx2 = random.randint(0,len(y_test))
plt.imshow(x_test[idx2,:])
plt.show()

y_pred = model.predict(x_test[idx2,:].reshape(1,100,100,3))
y_pred = y_pred > 0.5

if(y_pred==0):
    pred = 'dog'
else:
    pred = 'cat'

print("Our model says it is a",pred)
```


```
1/1 [==============================] - 0s 28ms/step
Our model says it is a dog
```

In [15]:
```
score = model.evaluate(x_test, y_test, verbose = 0 )
print("Test Score: ", score[0])
print("Test accuracy: ", score[1])
```

```
Test Score:  1.0204460620880127
Test accuracy:  0.7099999785423279
```

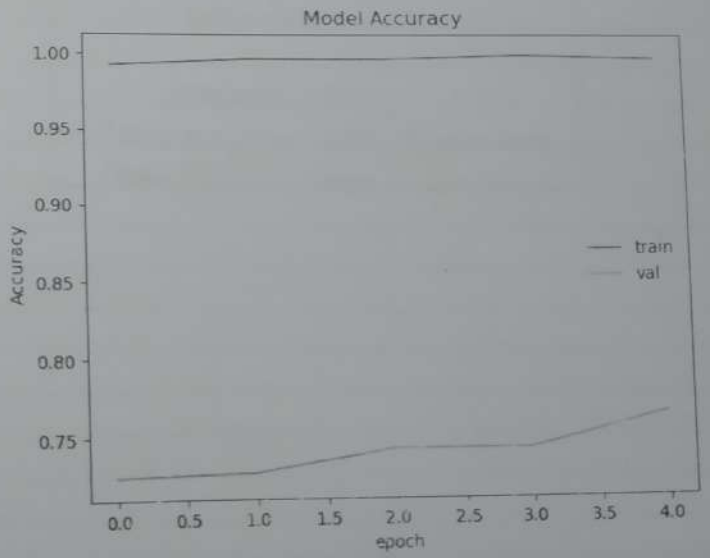
In [16]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 98, 98, 256)       7168

batch_normalization (BatchN  (None, 98, 98, 256)       1024
ormalization)

max_pooling2d (MaxPooling2D  (None, 24, 24, 256)       0
)

conv2d_1 (Conv2D)            (None, 22, 22, 128)       295040

batch_normalization_1 (Batc  (None, 22, 22, 128)       512
hNormalization)

max_pooling2d_1 (MaxPooling  (None, 11, 11, 128)       0
2D)

conv2d_2 (Conv2D)            (None, 9, 9, 64)          73792

batch_normalization_2 (Batc  (None, 9, 9, 64)          256
hNormalization)

max_pooling2d_2 (MaxPooling  (None, 4, 4, 64)          0
2D)

flatten (Flatten)            (None, 1024)              0

dense (Dense)                (None, 128)               131200

dropout (Dropout)            (None, 128)               0

dense_1 (Dense)              (None, 1)                 129

=================================================================
Total params: 509,121
Trainable params: 508,225
Non-trainable params: 896
_________________________________________________________________
```

In [17]:
```
val = model.fit(x_train,y_train, epochs=5,validation_data=(x_test,y_test),batch_s
```

```
Epoch 1/5
10/10 [==============================] - 61s 6s/step - loss: 0.0246 - accuracy:
0.9925 - val_loss: 0.9134 - val_accuracy: 0.7250
Epoch 2/5
10/10 [==============================] - 59s 6s/step - loss: 0.0173 - accuracy:
0.9970 - val_loss: 0.8590 - val_accuracy: 0.7275
Epoch 3/5
10/10 [==============================] - 58s 6s/step - loss: 0.0139 - accuracy:
0.9970 - val_loss: 0.8365 - val_accuracy: 0.7425
Epoch 4/5
10/10 [==============================] - 59s 6s/step - loss: 0.0101 - accuracy:
0.9995 - val_loss: 0.8777 - val_accuracy: 0.7425
Epoch 5/5
10/10 [==============================] - 59s 6s/step - loss: 0.0116 - accuracy:
0.9975 - val_loss: 0.8869 - val_accuracy: 0.7650
```

In [18]:
```
plt.title("Model Accuracy")
plt.ylabel("Accuracy")
plt.xlabel('epoch')
plt.plot(val.history['accuracy'])
plt.plot(val.history['val_accuracy'])
plt.legend(['train','val'])
plt.show()
```



Name: AKHIL. A
ROLL NO: 004
ASSIGNMENT NO: 4
SUBJECT : LP-IV (DL)

2022-23

NBN330E - 17
DOP: 29 Aug 2022 DOC: 12 Sept 2022
Marks: 10/10 Sign:

AIM: Use Autoencoder to implement anomaly detection. Build the model by using
a. Import required libraries.
b. Upload / acces 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.

OBJECTIVE: To learn about Autoencoders and developing autoencoders for anomaly detection.

Infrastructure: Computer / Laptop / Virtual Machine

Software used: Jupyter Notebook / Google Colab, Tensorflow, Keras

THEORY

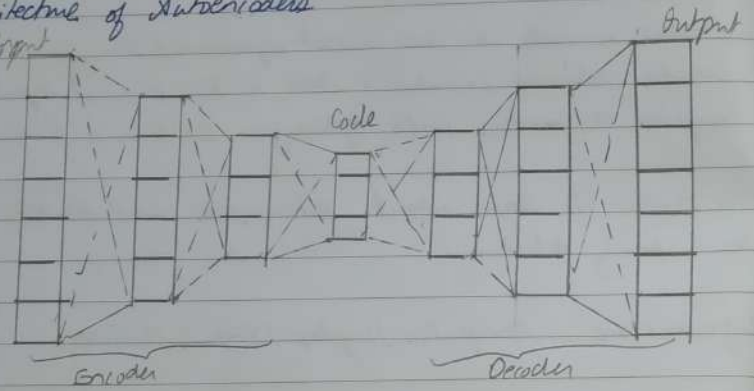
AUTO ENCODER

Autoencoder are artifical neural network capable of learning efficient representation of the input data, called codings, without using supervision. These codings have typically a much lower dimensionality than the input data, making autoencoders useful for dimensionality reduction and compression. These codings, the code is a compact "summary" or "compression" of the input.

Autoencoders act as powerful feature detectors and can be used for unsupervised pre-training of deep neural networks. Similiarly, they are capable of randomly generating new data that looks very similar to the training data. For example, you can train an encoder

on picture of faces and it would then be able to gener faces.

Architecture of Autoencoders



An autoencoder consists of 3 components:

i. Encoder: It compresses the input into a latent space repres The encoder layer encodes the input image as a compres representation in a reduced dimension. The compressed image is the distorted version of the original image.

ii. Code: This is the compressed input (from encoder) which to decoder for reconstructing the original input later.

iii. Decoder: It decodes the encoded output in form of code, original input. The decoded output is a lossy reconstruc original input. The goal is to get an output as iden was the input.

The layer between the encoder and decoder i.e. the code known as Bottleneck. This is a well-designed approach t

which aspects of the observed data are relevant information and what aspects can be discarded.

Parameters used for training an autoencoders:
Four parameters are:-

i. Code size: It is a number of nodes in the middle layer. Smaller size results in more compression and it may be difficult to make the size smaller beyond a certain limit to get satisfactorily results.

ii. Number of Layers: The autoencoders can be as deep as you like. They are very similar to an ANN, you only need to decide how many layers autoencoders should have.

iii. Number of nodes per layers: Number of nodes per layer decreases with each subsequent layer of the encoder and increases back in the decoder. Also the decoder is usually symmetric to the encoder in terms of layer structure.

iv. Loss function: We use either MSE (Mean Square Error) or Binary Cross entropy as the loss function. If the input values are ~~in the range~~ in the range [0,1] then you typically use cross-entropy, otherwise you use the mean square error.

Implementations:

i. Load the necessary libraries.
ii. Import the dataset from the respective library.
iii. Shape the data as per your needs.
iv. Encode the input data in latent representation.
v. Decode the output of the encoder to convert it back to original input.
vi. Use the models with Optimizers, Loss, and Evaluation Metrics.

Conclusion:
We learnt how to detect anomaly using ~~encoders~~ autoencoders

Name:- Akhil A
Roll No:- 04
PRN No:- 72036135E
Class:- BE[IT]
Subject:- Deep Learning

In [1]:
```
import matplotlib.pyplot as plt
import pandas as pd
import tensorflow as tf
import seaborn as sns
from tensorflow.keras.models import Model
from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.model_selection import train_test_split
from keras import Sequential
sns.set()
import numpy as np
from tensorflow.keras import layers, losses
```

In [2]:
```
df = pd.read_csv('http://storage.googleapis.com/download.tensorflow.org/data/ecg.
raw_data = df.values
df.head()
```

Out[2]:

	0	1	2	3	4	5	6	7	8
0	-0.112522	-2.827204	-3.773897	-4.349751	-4.376041	-3.474986	-2.181408	-1.818286	-1.250522
1	-1.100878	-3.996840	-4.285843	-4.506579	-4.022377	-3.234368	-1.566126	-0.992258	-0.754680
2	-0.567088	-2.593450	-3.874230	-4.584095	-4.187449	-3.151462	-1.742940	-1.490659	-1.183580
3	0.490473	-1.914407	-3.616364	-4.318823	-4.268016	-3.881110	-2.993280	-1.871131	-1.333884
4	0.800232	-0.874252	-2.384761	-3.973292	-4.338224	-3.802422	-2.534510	-1.783423	-1.594450

5 rows × 141 columns

In [3]:
```
labels = raw_data[:, -1]

data = raw_data[:, 0:-1]
```

In [4]:
```
pd.Series(labels).value_counts()
```

Out[4]:
```
1.0    2919
0.0    2079
dtype: int64
```

In [5]:
```
train_data,test_data, train_labels,test_labels = train_test_split(
    data, labels, test_size = 0.2, random_state=21
)
```

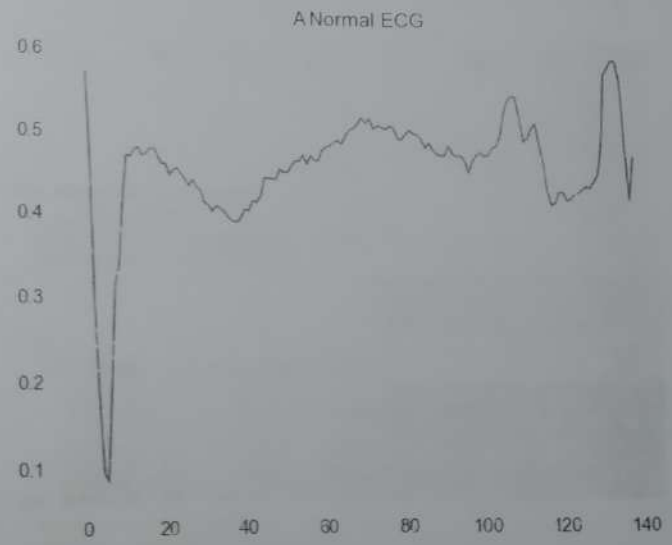
In [6]:
```
min = np.min(train_data)
max = np.max(train_data)

train_data = ( train_data - min ) / ( max - min )
test_data = ( test_data - min ) / ( max - min )
```

In [7]:
```
train_labels = train_labels.astype(bool)
test_labels = test_labels.astype(bool)


normal_train_data = train_data[train_labels]
normal_test_data = test_data[test_labels]

anamalous_train_data = train_data[~train_labels]
anamalous_test_data = test_data[~test_labels]
```

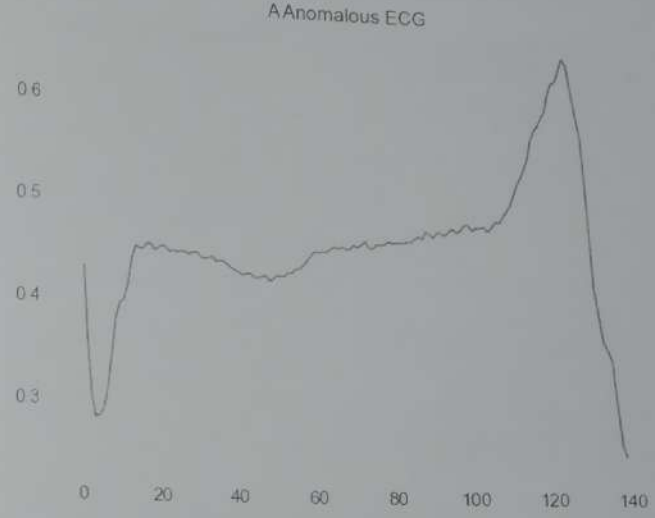
In [8]:
```
~train_labels
```

Out[8]:
```
array([False, False, False, ..., False, False, False])
```

In [9]:
```
plt.grid()
plt.plot(np.arange(140),normal_train_data[0])
plt.title('A Normal ECG')
plt.show()
```



In [10]:
```
plt.grid()
plt.plot(np.arange(140),anamalous_train_data[0])
plt.title('A Anomalous ECG')
plt.show()
```



In [11]:
```
class AnomalyDetector(Model):
  def __init__(self):
    super(AnomalyDetector, self).__init__()
    self.encoder = Sequential([
                                        layers.Dense(32, activation='relu'),
                                        layers.Dense(16, activation='relu'),
                                        layers.Dense(8, activation='relu')
    ])
    self.decoder = tf.keras.Sequential([
                                        layers.Dense(16, activation='relu'),
                                        layers.Dense(32, activation='relu'),
                                        layers.Dense(140, activation='sigmoid')
    ])

  def call(self, x):
    encoded = self.encoder(x)
    decoded = self.decoder(encoded)
    return decoded

autoencoder = AnomalyDetector()
```

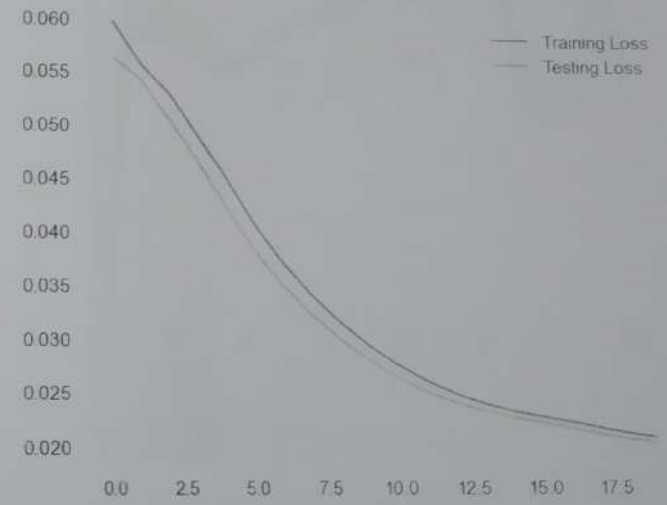
In [12]: autoencoder.compile(optimizer='adam', loss='mae')

```
In [13]: history = autoencoder.fit(normal_train_data, normal_train_data,
                                   epochs = 20,
                                   batch_size=512,
                                   validation_data=(normal_test_data, normal_test_data),
                                   shuffle=True)
Epoch 1/20
5/5 [==============================] - 1s 45ms/step - loss: 0.0598 - val_loss:
0.0565
Epoch 2/20
5/5 [==============================] - 0s 10ms/step - loss: 0.0559 - val_loss:
0.0545
Epoch 3/20
5/5 [==============================] - 0s 10ms/step - loss: 0.0533 - val_loss:
0.0508
Epoch 4/20
5/5 [==============================] - 0s 10ms/step - loss: 0.0494 - val_loss:
0.0470
Epoch 5/20
5/5 [==============================] - 0s 9ms/step - loss: 0.0456 - val_loss:
0.0428
Epoch 6/20
5/5 [==============================] - 0s 10ms/step - loss: 0.0413 - val_loss:
0.0387
Epoch 7/20
5/5 [==============================] - 0s 9ms/step - loss: 0.0376 - val_loss:
0.0354
Epoch 8/20
5/5 [==============================] - 0s 10ms/step - loss: 0.0344 - val_loss:
0.0326
Epoch 9/20
5/5 [==============================] - 0s 10ms/step - loss: 0.0319 - val_loss:
0.0303
Epoch 10/20
5/5 [==============================] - 0s 10ms/step - loss: 0.0297 - val_loss:
0.0283
Epoch 11/20
5/5 [==============================] - 0s 10ms/step - loss: 0.0279 - val_loss:
0.0267
Epoch 12/20
5/5 [==============================] - 0s 9ms/step - loss: 0.0264 - val_loss:
0.0253
Epoch 13/20
5/5 [==============================] - 0s 10ms/step - loss: 0.0251 - val_loss:
0.0243
Epoch 14/20
5/5 [==============================] - 0s 10ms/step - loss: 0.0242 - val_loss:
0.0235
Epoch 15/20
5/5 [==============================] - 0s 10ms/step - loss: 0.0235 - val_loss:
0.0229
Epoch 16/20
5/5 [==============================] - 0s 10ms/step - loss: 0.0230 - val_loss:
0.0224
Epoch 17/20
5/5 [==============================] - 0s 10ms/step - loss: 0.0225 - val_loss:
0.0219
```

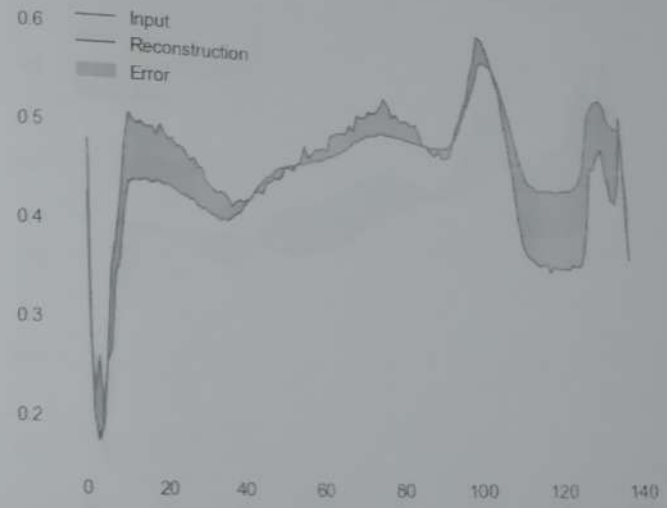
```
Epoch 18/20
5/5 [==============================] - 0s 9ms/step - loss: 0.0220 - val_loss:
0.0214
Epoch 19/20
5/5 [==============================] - 0s 10ms/step - loss: 0.0215 - val_loss:
0.0209
Epoch 20/20
5/5 [==============================] - 0s 9ms/step - loss: 0.0210 - val_loss:
0.0206
```

In [14]:
```
plt.plot(history.history['loss'],label='Training Loss')
plt.plot(history.history['val_loss'],label='Testing Loss')
plt.legend()
```

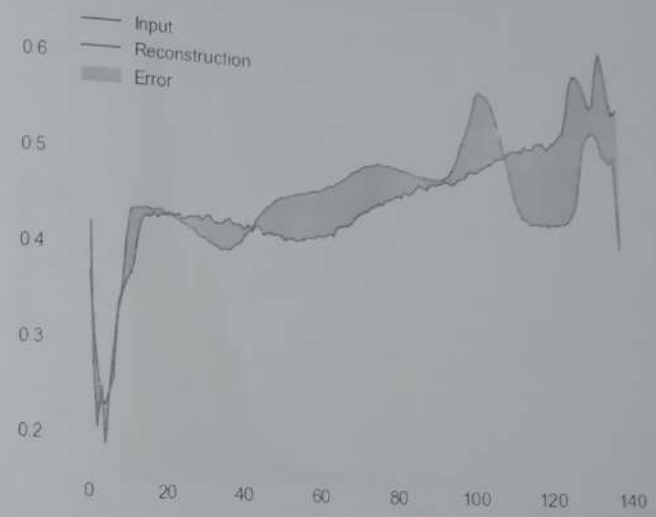
Out[14]:
```
<matplotlib.legend.Legend at 0x23e12fef790>
```



In [15]:
```
encoded_image = autoencoder.encoder(normal_test_data).numpy()
decoded_image = autoencoder.decoder(encoded_image).numpy()

plt.plot(normal_test_data[0],'b')
plt.plot(decoded_image[0],'r')
plt.fill_between(np.arange(140),decoded_image[0],normal_test_data[0],color='light
plt.legend(labels=['Input',"Reconstruction","Error"])
plt.show()
```

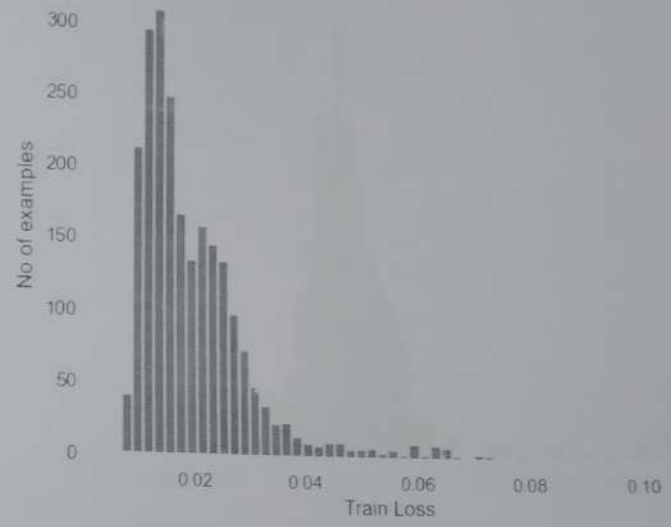


In [16]:
```
encoded_image = autoencoder.encoder(anamalous_test_data).numpy()
decoded_image = autoencoder.decoder(encoded_image).numpy()

plt.plot(anamalous_test_data[0],'b')
plt.plot(decoded_image[0],'r')
plt.fill_between(np.arange(140),decoded_image[0],anamalous_test_data[0],color='l
plt.legend(labels=['Input',"Reconstruction","Error"])
plt.show()
```



In [17]:
```
reconstructions = autoencoder.predict(normal_train_data)
train_loss = tf.keras.losses.mae(reconstructions, normal_train_data)

plt.hist(train_loss[None,:],bins=50)
plt.xlabel("Train Loss")
plt.ylabel("No of examples")
plt.show()
```

```
74/74 [==============================] - 0s 983us/step
```

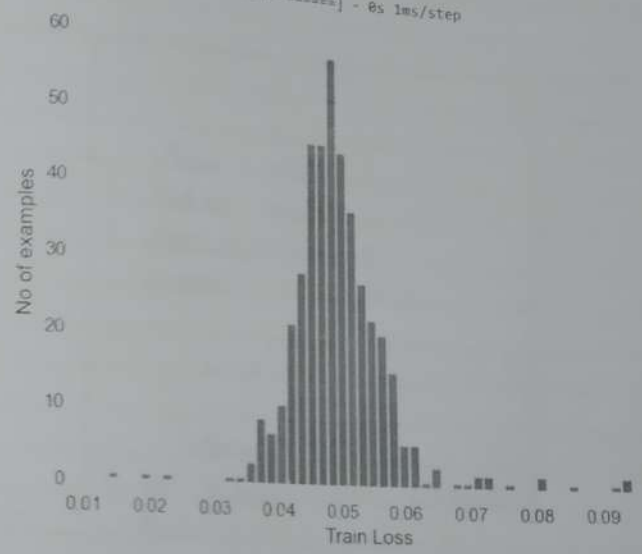


In [18]:
```
threshold = np.mean(train_loss) + np.std(train_loss)
print("Threshold: ",threshold)
```

```
Threshold:  0.03308283181023525
```

In [19]:
```
reconstructions = autoencoder.predict(anamalous_test_data)
test_loss = tf.keras.losses.mae(reconstructions, anamalous_test_data)

plt.hist(test_loss[None,:],bins=50)
plt.xlabel("Train loss")
plt.ylabel("No of examples")
plt.show()
```

```
14/14 [==============================] - 0s 1ms/step
```



In [20]:
```
def predict(model,data,threshold):
  reconstructions = model(data)
  loss = tf.keras.losses.mae(reconstructions,data)
  return tf.math.less(loss, threshold)

def print_stats(predictions, labels):
  print("Accuracy = {}".format(accuracy_score(labels,preds)))
  print("Precision = {}".format(precision_score(labels,preds)))
  print("Recall = {}".format(recall_score(labels,preds)))
```

In [21]:
```
preds = predict(autoencoder, test_data, threshold)
print_stats(preds, test_labels)
```

```
Accuracy = 0.945
Precision = 0.9922027290448343
Recall = 0.9089285714285714
```

Name : ANJIL A
Roll No : 004
ASSIGNMENT NO : 5
SUBJECT : LP-IV (DL)

2022-23

NBMSDE-II
DOP : 12 Sep 2022 DOC : 19 Sep 2022
Marks : 10/10 Sign :

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

OBJECTIVE: To learn and understand continous bag of words model.

Infrastructure: Computer / Laptop / Virtual Machine

Software used: Jupyter Notebook / Google Colab, Tensorflow, Keras

THEORY:
What is the CBOW Model?
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. Let us consider an example for understa
Consider the ~~sent~~ sentence: "It is a pleasant day" and the word "pleasant" goes as an input to the neural network. We are trying to predict the word 'day' here. We will use the one-hot encoding for the input words and measure the error rates with the one-hot encoded target word. Doing this will help us predict the output ~~with~~ based on the word with least error.

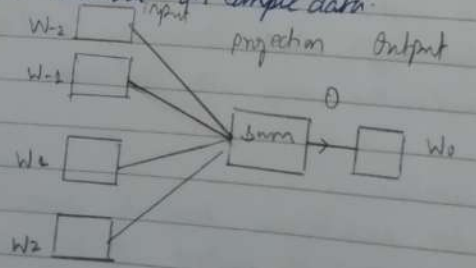
The Model Architecture
- The CBOW model architecture tries to predict the target word by trying to understand the context of the surrounding words.

Consider the same sentence as above, "It is a pleasant day" model converts this sentence into words pairs in the form (context word, target word). The user will have to set the window size. If the window for the context word is 2 then the pairs would look like this: ([it, a], is), ([is, pleasant], a), (pleasant). With these word pairs, the model tries to predict the word considered the context words

If we have 4 context words used for predicting one target word the input layer will be in the form of four 1 x W input vectors. These input vector will be passed to the hidden layer enters the sum layers where an element-wise summation is performed on the vectors before a final activation is performed and the output is obtained.

Implementation of the CBOW model

- Import the libraries and read our dataset.
- For the implementation of this model, we will use a sample text data.
- Generate function that create window sizes and pairs of words
- Build neural network on sample data.



Conclusion:

We saw what is CBOW model and how it works. These can be used for text recognition, speech to text conversion, etc.

Name:- Akhil A
Roll No:- 04
PRN No:- 72036135E
Class:- BE[IT]
Subject:- DL

In [2]:
```
import numpy as np
import tensorflow as tf
import keras.backend as K
from tensorflow.keras.models import Sequential
from keras.layers import Dense, Embedding, Lambda
from keras.utils import np_utils
from keras.preprocessing import sequence
from keras.preprocessing.text import Tokenizer
import gensim
```

In [4]:
```
data=open('covid.txt','r')
corona_data = [text for text in data if text.count(' ') >= 2]
vectorize = Tokenizer()
vectorize.fit_on_texts(corona_data)
corona_data = vectorize.texts_to_sequences(corona_data)
total_vocab = sum(len(s) for s in corona_data)
word_count = len(vectorize.word_index) + 1
window_size = 2
```

In [11]:
```
def 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 < t
            target.append(word)
            contextual = sequence.pad_sequences(context_word, total_length=total_
            final_target = np_utils.to_categorical(target, total_vocab)
            yield(contextual, final_target)
```

In [15]:
```
model = Sequential()
model.add(Embedding(input_dim=total_vocab, output_dim=100, input_len
model.add(Lambda(lambda x: K.mean(x, axis=1), output_shape=(100,)))
model.add(Dense(total_vocab, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam')
for i in range(20):
  cost = 0
  for x, y in cbow_model(data, window_size, total_vocab):
      cost += model.train_on_batch(contextual, final_target)
  print(i, cost)
```

```
0 0
1 0
2 0
3 0
4 0
5 0
6 0
7 0
8 0
9 0
10 0
11 0
12 0
13 0
14 0
15 0
16 0
17 0
18 0
19 0
```

In [16]:
```
dimensions=100
vect_file = open('vectors.txt' ,'w')
vect_file.write('{} {}\n'.format(101,dimensions))
```

Out[16]:
```
8
```

In [17]:
```
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))
vect_file.close()
```

```
cbow_output = gensim.models.KeyedVectors.load_word2vec_format('vectors.txt', bin=
cbow_output.most_similar(positive=['virus'])
```

Out[17]:
```
[('generated', 0.23279836773872375),
 ('19', 0.22727039456367493),
 ('present', 0.17593255639076233),
 ('appear', 0.16557843983173370),
 ('not', 0.16481697559356690),
 ('between', 0.14577879011631012),
 ('days', 0.13453739881515503),
 ('the', 0.13396610319614410),
 ('does', 0.11900028586387634),
 ('pre', 0.11821701377630234)]
```

In [ ]:

Name: ANJIL. A
ROLL NO : 004
ASSIGNMENT NO : 6
SUBJECT : LP-IV (OL)

2022-23

NBNSIOE-21

DOP: 29 Sep 2022 DOC: 03 Oct 2022
Marks: 10/10 Sign

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.

Objective: To load a pre-trained model and improve its performance by Transfer Learning architecture.

Infrastructure: Computer / Laptop / Virtual Machine

Software used: Jupyter Notebook / Google Colab, Tensorflow, Keras

THEORY

What is Transfer Learning?

Transfer learning generally refers to a process where a model trained on one problem is used in some way on a second related problem. One or more layers from the trained model are used in a new model trained on the problem of interest. Transfer learning has the benefit of decreasing the training time for a neural network model and can result in lower generalization error.

The weights in re-used layers may be used as the starting point for the training process and adapted in response to the new problem. This usage treats transfer learning as a type of weight

initialization scheme. This may be useful when the first related problem has a lot more labelled data that the problem of interest and the similarity in the structure of the problem may be useful in both contexts.

How to use pre-trained models

Some of these useage patterns as follows:

i. Classifier: The pre-trained model is used directly to classify new images.

ii. Standalone Feature Extractor: The pre-trained model, or some portion of the model, is used to pre-process images and extract relevant features.

iii. Integrated Feature Extractor: The pre-trained model, or some portion of the model is integrated into a new model, but layers of the pre-trained model are frozen during training.

iv. Weight Initialization: The pre-trained, or some portion of a model, is integrated into a new model, and the layers of the pre-trained model are trained in concert with the new model.

It may not be clear as to which usage of the pre-trained model may yeild the best results on your new computer vision task. Therefore some experimentation may be required.

Ways to Fine tune the model

i. Feature extraction: We can use a pre-trained model as a feature extraction mechanism. What we can do is that we can remove the output layer and then use the entire network as a fixed feature extractor for the new data set.

ii. Use the architecture of the pre-trained model: What we can do is that we use architecture of the model while we initialize all the weights randomly and train the model according to our dataset again.

iii. Train some layers while freeze others: Another way to use a pre-trained model is to train it partially. What we can do it is we keep the weights of initial layers of the model frozen while we retrain only the higher layers. We can try and test as to how many layers are to be frozen and how many to be trained.

Building A Deep Learning-Based Object Detection Model

Training a performing deep learning model for object detection takes a lot of data and computing power. To facilitate the development, we can use transfer learning by fining tuning models pre-trained based on other relevant datasets.

Since there are multiple backend logistics such as paths and hyperparameters to take care of when training a full-scale deep learning model, we can create a central dictionary to store these configuration parameters, including setting up the different paths, installing relevant libraries, and downloading the pre-trained models.

Conclusion:

We concluded from the experiment, how to develop a model for a specific application with the help of transfer learning architecture in deep learning.

Assignment No 6

November 9, 2022

Name: Akhil A Roll No.: 004 PRN: 72036135E Subject: LP-IV(DL)

[1]:
```
import tensorflow as tf
import numpy as np
import cv2
import PIL.Image as Image
import os
import matplotlib.pyplot as plt
import tensorflow_hub as hub
import pathlib
```

[2]:
```
Image_Shape = (224,224)
```

[3]:
```
URL_dataset = "https://storage.googleapis.com/download.tensorflow.org/
↪example_images/flower_photos.tgz"
```

[4]:
```
data_dir = tf.keras.utils.get_file(origin=URL_dataset,
                                   fname='flower_photos',
                                   untar=True)
data_dir = pathlib.Path(data_dir)
```

```
Downloading data from https://storage.googleapis.com/download.tensorflow.org/exa
mple_images/flower_photos.tgz
228813984/228813984 [==============================] - 1s 0us/step
```

[5]:
```
# Total images
image_count = len(list(data_dir.glob('*/*.jpg')))
print(image_count)
```

```
3670
```

0.0.1 Making the different classes

[6]:
```
flowers_images_dict = {
    "daisy" : list(data_dir.glob('daisy/*')),
    "dandelion" : list(data_dir.glob('dandelion/*')),
    "roses" : list(data_dir.glob('roses/*')),
    "sunflowers" : list(data_dir.glob('sunflowers/*')),
    "tulips" : list(data_dir.glob('tulips/*'))
```

```
}
```

```
flowers_labels_dict= {
    "daisy" : 0,
    "dandelion" : 1,
     "roses" : 2,
    "sunflowers" : 3,
    "tulips" : 4
}
```

0.0.2 Reshaping Images

```
X, Y = [],[]

for flower_name, images in flowers_images_dict.items():
  for image in images:
    img = cv2.imread(str(image))
    resized_img = cv2.resize(img, Image_Shape)
    X.append(resized_img)
    Y.append(flowers_labels_dict[flower_name])
```

```
X = np.array(X)
y = np.array(Y)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

```
X_train_scaled = X_train / 255
X_test_scaled = X_test / 255
```

0.0.3 Pretrained Model

```
tf_model = "https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/4"
```

```
classifier = tf.keras.Sequential([
    hub.KerasLayer(tf_model,input_shape=(224,224,3), trainable=False),
    tf.keras.layers.Dense(len(flowers_labels_dict), activation="softmax")
])
classifier.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
keras_layer (KerasLayer)     (None, 1280)              2257984

dense (Dense)                (None, 5)                 6405
```

```
Total params: 2,264,389
Trainable params: 6,405
Non-trainable params: 2,257,984
```

```
classifier.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=["accuracy"]
)
```

```
classifier.fit(X_train_scaled, y_train,epochs=5)
```

```
Epoch 1/5
/usr/local/lib/python3.7/dist-packages/tensorflow/python/util/dispatch.py:1082:
UserWarning: "`sparse_categorical_crossentropy` received `from_logits=True`, but
the `output` argument was produced by a sigmoid or softmax activation and thus
does not represent logits. Was this intended?"
  return dispatch_target(*args, **kwargs)
86/86 [==============================] - 15s 42ms/step - loss: 0.8199 -
accuracy: 0.6962
Epoch 2/5
86/86 [==============================] - 4s 45ms/step - loss: 0.4254 - accuracy:
0.8503
Epoch 3/5
86/86 [==============================] - 4s 46ms/step - loss: 0.3299 - accuracy:
0.8950
Epoch 4/5
86/86 [==============================] - 4s 43ms/step - loss: 0.2745 - accuracy:
0.9179
Epoch 5/5
86/86 [==============================] - 4s 47ms/step - loss: 0.2363 - accuracy:
0.9320
```

```
<keras.callbacks.History at 0x7f76500871d0>
```

```
classifier.evaluate(X_test_scaled, y_test)
```

```
29/29 [==============================] - 2s 55ms/step - loss: 0.3484 - accuracy:
0.8725
```

```
[0.348428338766098, 0.8725489974021912]
```

0.0.4 Prediction

```
from PIL import Image
img = Image.open("360_F_105573812_cvD4P5jo6tMPhZULX324qUYFbNpXlisD.jpg")
img
```

```
img = tf.keras.preprocessing.image.img_to_array(img.resize(Image_Shape))
img = np.array([img])
```

```
res = classifier.predict(img)
```

```
1/1 [==============================] - 1s 737ms/step
```

```
print("The prediction is : {}".format(list(flowers_labels_dict.keys())[np.
↪argmax(res)]))
```

```
The prediction is : roses
```