# 414447: LABORATORY PRACTICES IV

BEIT (2019 Course)

Semester - I

| Teaching Scheme | | Examination Scheme | |
|---|---|---|---|
| Practical : | 2 Hrs. / Week | Term work : | 25 Marks |
| | | Practical | 25 Marks |


LABORATORY MANUAL V 3.0

DEPARTMENT OF INFORMATION TECHNOLOGY

**Sinhgad Institute of Technology, Lonavala**

2022-2023

## Vision and Mission of Institute

**VISION**

उत्तमपुरुषान् उत्तमाभियंतृन् निर्मातुं कटीबध्दा: वयम्।

We are committed to produce not only good engineers but good human beings, also.

**MISSION**

- We believe in and work for the holistic development of students and teachers.
- We strive to achieve this by imbibing a unique value system, transparent work culture, excellent academic and physical environment conducive to learning, creativity and technology transfer.

## Vision and Mission of the Department

**VISION**

The departments of IT will practice teaching-learning methodologies to achieve recognition at national and international level in engineering education specifically for Information and Communication Technology (ICT) sector for dynamic needs of local and global industry and professional community.

**MISSION**

The department offers an inviting, nurturing, and challenging environment for self-learning, which is responsive to the intellectual, social and cultural needs of a diverse learning community. The success of department is reflected in well-being of its alumni, who are known for their leadership, adaptability and commitment to high professional standards.

## PROGRAM EDUCATIONAL OBJECTIVES

The students of Information Technology course after passing out will

1) Graduates of the program will possess strong fundamental concepts in mathematics, science, engineering and Technology to address technological challenges.

2) Possess knowledge and skills in the field of Computer Science & Engineering and Information Technology for analyzing, designing and implementing complex engineering problems of any domain with innovative approaches.

3) Possess an attitude and aptitude for research, entrepreneurship and higher studies in the field of Computer Science & Engineering and Information Technology.

4) Have commitment to ethical practices, societal contributions through communities and life-long learning.

5) Possess better communication, presentation, time management and team work skills leading to responsible & competent professionals and will be able to address challenges in the field of IT at global level.

## PROGRAM OUTCOMES

The students in the Information Technology course will attain:

a. an ability to apply knowledge of computing, mathematics including discrete mathematics as well as probability and statistics, science, and engineering and technology;
b. an ability to define a problem and provide a systematic solution with the help of conducting experiments, as well as analyzing and interpreting the data;
c. an ability to design, implement, and evaluate a software or a software/hardware system, component, or process to meet desired needs within realistic constraints;
d. an ability to identify, formulate, and provide systematic solutions to complex engineering problems;
e. an ability to use the techniques, skills, and modern engineering technologies tools, standard processes necessary for practice as a IT professional;
f. an ability to apply mathematical foundations, algorithmic principles, and computer science theory in the modeling and design of computer-based systems with necessary constraints and assumptions;
g. an ability to analyze the local and global impact of computing on individuals, organizations and society;
h. an ability to understand professional, ethical, legal, security and social issues and responsibilities;
i. an ability to function effectively as an individual or as a team member to accomplish a desired goal(s);
j. an ability to engage in life-long learning and continuing professional development to cope up with fast changes in the technologies/tools with the help of electives, professional organizations and extra-curricular activities;
k. an ability to communicate effectively in engineering community at large by means of effective presentations, report writing, paper publications, demonstrations;
l. an ability to understand engineering, management, financial aspects, performance, optimizations and time complexity necessary for professional practice;
m. an ability to apply design and development principles in the construction of software systems of varying complexity.

# Syllabus

| Savitribai Phule Pune University, Pune<br>Final Year Information Technology (2019 Course)<br>414447: Lab Practice IV | | |
|---|---|---|
| **Teaching Scheme:** | **Credit Scheme:** | **Examination Scheme:** |
| **Practical (PR):02 hrs/week** | **01 credits** | **PR: 25 Marks**<br>**TW: 25 Marks** |

**Prerequisites:** Python programming language

**Course Objectives:**

The objective of the course is

**1.** To be able to formulate deep learning problems corresponding to different applications.

**2.** To be able to apply deep learning algorithms to solve problems of moderate complexity.

**3.** To apply the algorithms to a real-world problem, optimize the models learned and report on the expected accuracy that can be achieved by applying the models.

**Course Outcomes:**

On completion of the course, students will be able to-

**CO1.** Learn and Use various Deep Learning tools and packages.

**CO2.** Build and train a deep Neural Network models for use in various applications.

**CO3.** Apply Deep Learning techniques like CNN, RNN Auto encoders to solve real word Problems.

**CO4.** Evaluate the performance of the model build using Deep Learning.

**Guidelines for Instructor's Manual**

The faculty member should prepare the laboratory manual for all the experiments, and it should be made available to students and laboratory instructor/assistant

**Guidelines for Student's Lab Journal**

**1.** Students should submit term work in the form of a handwritten journal based on a specified list of assignments.

**2.** Practical Examination will be based on the term work.

**3. Candidate is expected to know the theory involved in the experiment.**

**4.** The practical examination should be conducted if and only if the journal of the candidate is complete in all respects.

**Guidelines for Lab /TW Assessment**

**1.** Examiners will assess the term work based on performance of students considering the parameters such as timely conduction of practical assignment, methodology adopted for implementation of practical assignment, timely submission of assignment in the form of handwritten write-up along with results of implemented assignment, attendance etc.

**2.** Examiners will judge the understanding of the practical performed in the examination by asking some questions related to theory & implementation of experiments he/she has carried out.

**3.** Appropriate knowledge of usage of software and hardware related to the respective laboratory should be checked by the concerned faculty member.

# INDEX

# SCHEDULE

| Assign No. | Assignment Title | No. of Hrs. | Week No. |
|---|---|---|---|
| 1 | Study of Deep learning Packages: Tensorflow, Keras, Theano and PyTorch. Document the distinct features and functionality of the packages. | 4 | WK1,WK2 |
| 2 | Implementing Feedforward neural networks with Keras and TensorFlow<br>**a.** Import the necessary packages<br>**b.** Load the training and testing data (MNIST/CIFAR10)<br>**c.** Define the network architecture using Keras<br>**d.** Train the model using SGD<br>**e.** Evaluate the network<br>**f.** Plot the training loss and accuracy | 2 | WK3,WK4 |
| 3 | Build the Image classification model by dividing the model into following 4 stages:<br>**a. Loading and preprocessing the image data**<br>**b. Defining the model's architecture**<br>**c. Training the model**<br>**d. Estimating the model's performance** | 4 | WK5,WK6 |
| 4 | Use Autoencoder to implement anomaly detection. Build the model by using:<br>**a. Import required libraries**<br>**b. Upload / access the dataset**<br>**c. Encoder converts it into latent representation**<br>**d.** Decoder networks convert it back to the original input<br>**e.** Compile the models with Optimizer, Loss, and Evaluation Metrics | 2 | WK7,WK8 |
| 5 | Implement the Continuous Bag of Words (CBOW) Model. Stages can be:<br>**a.** Data preparation<br>**b.** Generate training data<br>**c.** Train model<br>**d.** Output | 2 | WK9,WK10 |
| 6 | Object detection using Transfer Learning of CNN architectures<br>**a.** Load in a pre-trained CNN model trained on a large dataset<br>**b.** Freeze parameters (weights) in model's lower convolutional layers<br>**c.** Add custom classifier with several layers of trainable parameters to model<br>**d.** Train classifier layers on training data available for task<br>**e.** Fine-tune hyper parameters and unfreeze more layers as needed | 2 | WK7,WK8 |

Sinhgad Institutes

*Name of the Student:* _____ _____

*Roll no:* _____

*CLASS: - B.E. IT* *Subject Name: - LP-IV Lab*

## Assignment No. 01

**** Study of Deep learning Packages: Tensorflow, Keras, Theano and PyTorch : CO1 ****

**Date of Performance:** / /2022

**Marks out of 10:**

## Assignment No: 1

### Title:

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

### Problem Statement

Study and installation of following Deep learning Packages:

i. **Tensor Flow**
ii. **Keras**
iii. **Theano**
iv. **PyTorch**

### Theory:

1) What is Deep learning?
2) What are various packages in python for supporting Machine Learning libraries andwhich are mainly used for Deep Learning ?
3) Compare Tensorflow / Keras/Theno and PyTorch on following points(make a table) :

i. Available Functionality
ii. GUI status
iii. Versions.
iv. Features
v. Compatibility with other environments.
vi. Specific Application domains.

4) Enlist the Models Datasets and pretrained models, Libraaries and Extensions , Tools related to Tensorflow also discuss any two case studies like (PayPal, Intel, Etc. ) related toTensor Flow. [Ref: https://www.tensorflow.org/about]

5) Explain the Keras Ecosystem. (keras tuner,keras NLP,kerasCV,Autokeras and Model optimization.) Also explain following concepts related to keras.

1. Developing sequential Model
2. Training and validation using the inbuilt functions
3. ParameterOptimization. [Ref: https://keras.io/]

6) Explain simple Theano program.

7) Explain PyTorch Tensors. And also explain Uber's Pyro, TesalaAutopilot. [https://pytorch.org/]

**Steps/ Algorithm**

Installation of Tensorflow on Ubntu:

1. Install the Python Development Environment:

You need to download Python, the PIP package, and a virtual environment. If thesepackages are already installed, you can skip this step.

You can download and install what is needed by visiting the following links: https://www.python.org/

https://pip.pypa.io/en/stable/installing/

https://docs.python.org/3/library/venv.html

To install these packages, run the following commands in the terminal:sudo apt update

sudo apt install python3-dev python3-pip python3-venv

1. Create a Virtual Environment

Navigate to the directory where you want to store your Python 3.0 virtual environment. It can be in your home directory, or any other directory where your user can read and write permissions.

mkdir tensorflow_filescd tensorflow_files

Now, you are inside the directory. Run the following command to create a virtual environment:

python3 -m venv virtualenv

The command above creates a directory named virtualenv. It contains a copy of the Python binary, the PIP package manager, the standard Python library, and other supportingfiles.

2. Activate the Virtual Environmentsource virtualenv/bin/activate

Once the environment is activated, the virtual environment' bin directory will be added to the beginning of the $PATH variable. Your shell's prompt will alter, and it will show thename of the virtual environment you are currently using, i.e. virtualenv.

3. Update PIP

pip install --upgrade pip

5. Install TensorFlow

The virtual environment is activated, and it's up and running. Now, it's time to install the TensorFlow package.

pip install -- upgrade TensorFlow

Installation of Keras on Ubntu :

Prerequisite : Python version 3.5 or above.

STEP 1: Install and Update Python3 and Pip

Skip this step if you already have Python3 and Pip on your machine. sudo apt install python3 python3.pip

sudo pip3 install – – upgrade pipSTEP 2: Upgrade Setuptools

pip3 install – – upgrade setuptoolsSTEP 3: Install TensorFlow

pip3 install tensorflow

Verify the installation was successful by checking the software package information: pip3 show tensorflow

STEP 4: Install Keras

pip3 install keras

Verify the installation by displaying the package information:

pip3 show keras

[https://phoenixnap.com/kb/how-to-install-keras-on-linux]Installation of Theano on Ubuntu:

**Step 1:** First of all, we will install Python3 on our Linux Machine. Use the following command in the terminal to install Python3.

sudo apt-get install python3

**Step 2**: Now, install the pip moduleudo apt install python3-pip

**Step 3**: Now, install the Theano

Verifying Theao package Installation on Linux using PIPpython3 -m pip show theano

**Installation of PyTorch**

First, check if you are using python's latest version or not.Because PyGame requires python 3.7 or a higher version

python3 – versionpip3 – version

pip3 install torch==1.8.1+cpu torchvision==0.9.1+cpu torchaudio==0.8.1 -f https://download.pytorch.org/whl/torch_stable.html

[Ref : https://www.geeksforgeeks.org/install-pytorch-on-linux/]

**Python 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

from sklearn.model_selection import train_test_split

2. For TheaonRequirements:Python3

Python3-pipNumPy SciPy

BLAS

**Sample Code with comments**

1. Tensorflow Test program:

```
import tensorflow as tf

print(tf.__version__)

2.1.0

print(tf.reduce_sum(tf.random.normal([1000, 1000])))

tf.Tensor(-505.04108, shape=(), dtype=float32)
```

2. Keras Test Program:

from tensorflow import keras

**from** keras **import** datasets

#

# Load MNIST data#

(train_images, train_labels), (test_images, test_labels) = datasets.mnist.load_data()

#

# Check the dataset loaded#

train_images.shape, test_images.shape

3. Theano test program

# Python program showing# addition of two scalars

```
# Addition of two scalarsimport numpy
import theano.tensor as T from theano import function

# Declaring two variablesx = T.dscalar('x')
y = T.dscalar('y')

# Summing up the two numbersz = x + y

# Converting it to a callable object
# so that it takes matrix as parametersf = function([x, y], z)
f(5, 7)
```

4. Test program for PyTorch

```
## The usual importsimport torch
import torch.nn as nn


## print out the pytorch version usedprint(torch.__version__)
```

## Conclusion:

Tensorflow , PyTorch,Keras and Theano all these packages are installed and ready for Deep learning applications . As per application domain and dataset we can choose the appropriate package and build required type of Neural Network.

Sinhgad Institutes

*Name of the Student:* _____

*Roll no:* _____

*CLASS: - B E. IT* *Subject Name: - LP-IV Lab*

## Assignment No. 02

**** Implementing Feedforward NN with Keras and TensorFlow: CO1 ****

**Date of Performance:** / /2022

**Marks out of 10:**

**Sign with Date:**

# Assignment No. 02

## Problem Statement

Implementation of Feed forward Neural Network with keras and tensorflow.

1. Import the necessary packages
2. Load the training and testing data(MNIST)
3. Define the network architecture using keras
4. Train the model using SGD
5. Evaluate the network
6. Plot the training loss and accuracy

## Solution Expected

Implement and train a feed-forward neural network (also known as an "MLP" for "multi-layer perceptron") on a dataset called MNIST and improve model generalization by achieving increased accuracy and descresesd loss where model gains good confidence with the prediction.

## Objectives to be achieved

1. Understand how to use Tensorflow Eager and Keras Layers to build neural network architecture.
2. Understand how a model is trained and evaluated.
3. Identify digits from images.
4. Our main goal is to train a neural network (using Keras) to obtain > *90%* accuracy on MNIST dataset.
5. Research at least 1 technique that can be used to improve model generalization.

## Methodology to be used

- Deep Learning
- Feed Forward Neural Network

## Justification with Theory/Literature

**Deep learning** has revolutionized the world of machine learning as more and more ML practitioners have adopted deep learning networks to solve real-world problems. Compared to the more traditional ML models, deep learning networks have been shown superior performance for many applications.

The first step toward using deep learning networks is to understand the working of a simple feedforward neural network we get started with how we can build our first neural network model using **Keras** running on top of the **Tensorflow** library.

TensorFlow is an open-source platform for machine learning. Keras is the high-level application programming interface (API) of TensorFlow. Using Keras, we can rapidly develop a prototype system and test it out. This is the first in a three-part series on using TensorFlow for supervised classification tasks.

### A Conceptual Diagram of the Neural Network

we'll build a supervised classification model that learns to identify digits from images. We'll use the well-known MNIST dataset to train and test our model. The MNIST dataset consists of 28-by-28 images of handwritten digits along with their corresponding labels.

We'll construct a neural network with one hidden layer to classify each digit image into its corresponding label. The figure below shows a conceptual diagram of the neural network we are about to build. The output layer consists of 10 units, where each unit corresponds to a different digit. Each unit computes a value that can be interpreted as the confidence value of its respective digit. The final classification is the digit with the maximum confidence value.

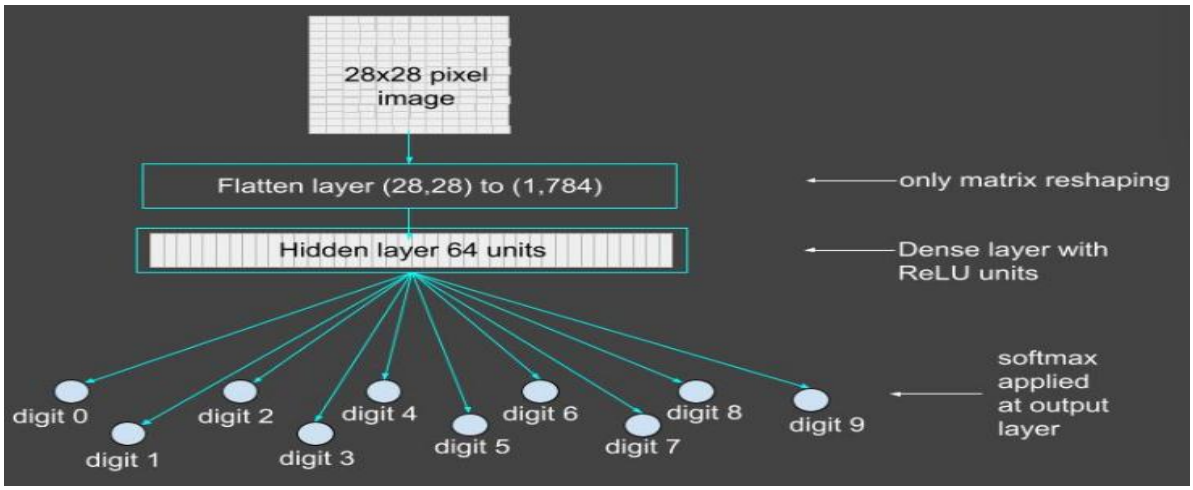



Figure 1: Conceptual diagram of the neural network. Each output unit corresponds to a digit and has its confidence value. The final classification is the digit with the maximum confidence value.

**TensorFlow and Keras Libraries**

If Keras and TensorFlow are not installed on your system, you can easily do so using pip or conda depending upon your Python environment.

```
pip install tensorflow
```

In the context of ML, a tensor is a multidimensional array, which in its simplest form is a scalar. Vectors and matrices are special cases of tensors. In TensorFlow, a tensor is a data structure. It is a multidimensional array composed of elements of the same type. Tensors are used to encapsulate all inputs and outputs to a deep learning network. The training dataset and each test example has to be cast as a tensor. All operations within the layers of the network are also performed on tensors.

**Layers in TensorFlow?**

You can build a fully connected feedforward neural network by stacking layers sequentially so that the output of one layer becomes the input to the next. In TensorFlow, layers are callable objects, which take tensors as input and generate outputs that are also tensors. Layers can contain weights and biases, which are both tuned during the training phase. We'll create a simple neural network from two layers:

1. Flatten layer
2. Dense layer

The Flatten Layer:

This layer flattens an input tensor without changing its values. Given a tensor of rank n, the Flatten layer reshapes it to a tensor of rank 2. The number of elements on the first axis remains unchanged. The elements of the input tensor's remaining axes are stacked together to form a single axis. We need this layer to create a vectorized version of each image for further input to the next layer.

The Dense Layer

The dense layer is the fully connected, feedforward layer of a neural network. It computes the weighted sum of the inputs, adds a bias, and passes the output through an activation function. We are using the ReLU activation function for this example. This function does not change any value greater than 0. The rest of the values are all set to 0.

The computations of this layer for the parameters shown in the code above are all illustrated in the figure below.

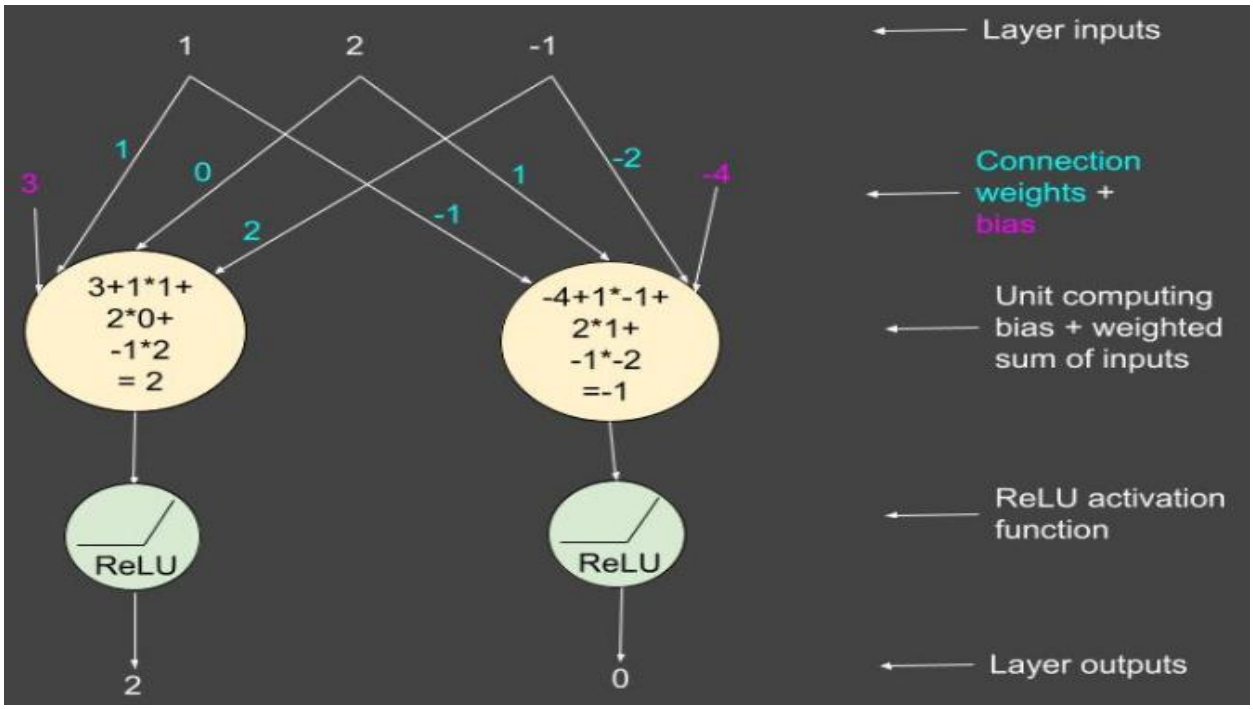



Figure 2: Hidden layer computations.

**Creating a Model in TensorFlow**

We are now ready to create a model of a simple neural network with one hidden layer. The simplest method is to use Sequential() with a list of all the layers you want to stack together. The code below creates a model and prints its summary. Note the use of relu as the activation function in the first dense layer and a softmax in the output layer. The softmax function normalizes all outputs to sum to 1 and ensures that they are in the range [0, 1].

```
model = tf.keras.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(64, activation='relu'),
  tf.keras.layers.Dense(10, activation='softmax')
```

**Compile the Model**

Next we compile the model. Here, we specify the optimizer and loss function of our model. The optimization algorithm determines how the connection weights are updated at each training step with respect to the given loss function. Because we have a multiclass classification problem, the loss function we'll use is categorical_crossentropy, coupled with the adam optimizer. You can

experiment with other optimizers too. The value of the metrics argument sets the parameter to monitor and record during the training phase.

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

**Train the Neural Network**

Now that the model is ready, it's time to train it. We'll load the dataset, train the model, and view the training process. Note that the outputs shown here will vary with every run of the program because of the stochastic nature of the algorithms involved.

**Load the Dataset**

The following code loads the training set and the test set of the MNIST data. It also prints the statistics of both sets. Because our model has 10 outputs, one for each digit, we need to convert the absolute image labels to categorical ones. The utils module in the Keras library provides the method to_categorical() for this conversion.

**Train the Model**

The fit() method of the model object trains the neural network. If you want to use a validation set during training, all you have to do is define the percentage of validation examples to be taken from the training set. The splitting of the training set into a train and validation set is automatically taken care of by the fit() method.

In the code below, the fit() method is called in 10 epochs.

**View the Training Process**

The fit() method returns a history object with detailed information regarding model training. The history attribute is a dictionary object.

To view the learning process, we can plot the accuracy and loss corresponding to different epochs for both the training and validation sets. The following code creates two graphs for these two metrics.

**The predict() method**

If you want to see the output of the network for one or more train/test examples, use the predict() method. The following example code prints the values of the output layer when the first test image is used as an input. It is a 10-dimensional vector of confidence values corresponding to each digit. The final classification of the image is the argmax() of this vector.

**The evaluate() method**

The evaluate() method computes the overall accuracy of the model on the dataset passed as an argument. The code snippet below prints the classification accuracy on both the training set and the test set.

# Code Snippets

## Importing Libraries

```
[1]  #import necessary libraries
     import tensorflow as tf
     from tensorflow import keras
```

```
[2]  import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     import random
     %matplotlib inline
```

## Loading & Preparing the data

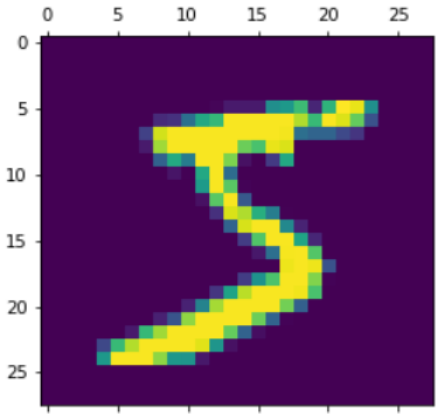
MNIST stands for Modified National Institute of Standards and Technology dataset. It is a dataset of 70,000 handwritten images.Each image is of 28*28 pixel i.e about 784 features. Each feature represent only one pixel intensity i.e from 0(white) to 255(black). This dataset is further devided into 60000 training and 10000 testing images.

```
#import dataset and split into train and test
mnist = tf.keras.datasets.mnist
(x_train,y_train),(x_test,y_test) = mnist.load_data()
```

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

```
#to see how first image looks
plt.matshow(x_train[0])
```

```
<matplotlib.image.AxesImage at 0x7f686e32ed10>
```




normalizing the images by scaling the pixel intensities to the range 0 to 1

```
[ ] #normalizing the images by scaling the pixel intensities to the range 0 to 1
    x_train = x_train/255
    x_test = x_test/255
```

```
x_train[0]
```

```
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.19215686, 0.93333333, 0.99215686,
        0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.99215686,
        0.99215686, 0.99215686, 0.98431373, 0.36470588, 0.32156863,
        0.32156863, 0.21960784, 0.15294118, 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.07058824, 0.85882353, 0.99215686,
        0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.77647059,
        0.71372549, 0.96862745, 0.94509804, 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.31372549, 0.61176471,
        0.41960784, 0.99215686, 0.99215686, 0.80392157, 0.04313725,
        0.        , 0.16862745, 0.60392157, 0.        , 0.        ,
```

✓ 0s completed at 2:22 PM

## Creating the model

The Relu function is one of the most popular activation function. It stands for"Rectified Linear Unit". Mathematically this function is defined as y=max(0,x). The relu function returns 0 if the input is negative and linear if the input is positive.

The softmax function is another activation function. It changes input values into values that reach from 0 to 1.

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

```
[ ] model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_2 (Flatten)         (None, 784)               0

 dense_4 (Dense)             (None, 128)               100480

 dense_5 (Dense)             (None, 10)                1290

=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
```

## Compile the model

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

## Train the model

```
[ ] history=model.fit(x_train,y_train,validation_data=(x_test,y_test),epochs=10)
```

```
Epoch 1/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.6254 - accuracy: 0.8444 - val_loss: 0.3584 - val_accuracy: 0.90
Epoch 2/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.3366 - accuracy: 0.9053 - val_loss: 0.3012 - val_accuracy: 0.91
Epoch 3/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.2889 - accuracy: 0.9188 - val_loss: 0.2641 - val_accuracy: 0.92
Epoch 4/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.2583 - accuracy: 0.9276 - val_loss: 0.2405 - val_accuracy: 0.93
Epoch 5/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.2352 - accuracy: 0.9345 - val_loss: 0.2215 - val_accuracy: 0.93
Epoch 6/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.2162 - accuracy: 0.9392 - val_loss: 0.2047 - val_accuracy: 0.94
Epoch 7/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.2004 - accuracy: 0.9437 - val_loss: 0.1907 - val_accuracy: 0.94
Epoch 8/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.1871 - accuracy: 0.9472 - val_loss: 0.1821 - val_accuracy: 0.94
Epoch 9/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.1755 - accuracy: 0.9505 - val_loss: 0.1709 - val_accuracy: 0.95
Epoch 10/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.1653 - accuracy: 0.9539 - val_loss: 0.1622 - val_accuracy: 0.95
```

✓ 0s completed at 2:22 PM

## Evaluate the model

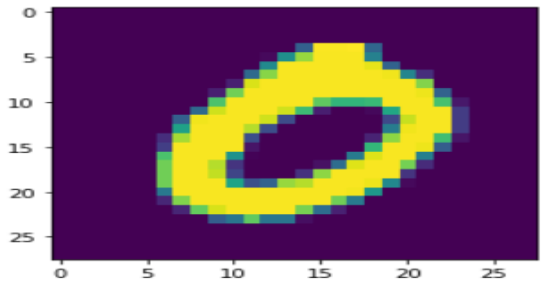
```
[ ] test_loss,test_acc=model.evaluate(x_test,y_test)
    print("Loss=%.3f" %test_loss)
    print("Accuracy=%.3f" %test_acc)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.1622 - accuracy: 0.9521
Loss=0.162
Accuracy=0.952
```

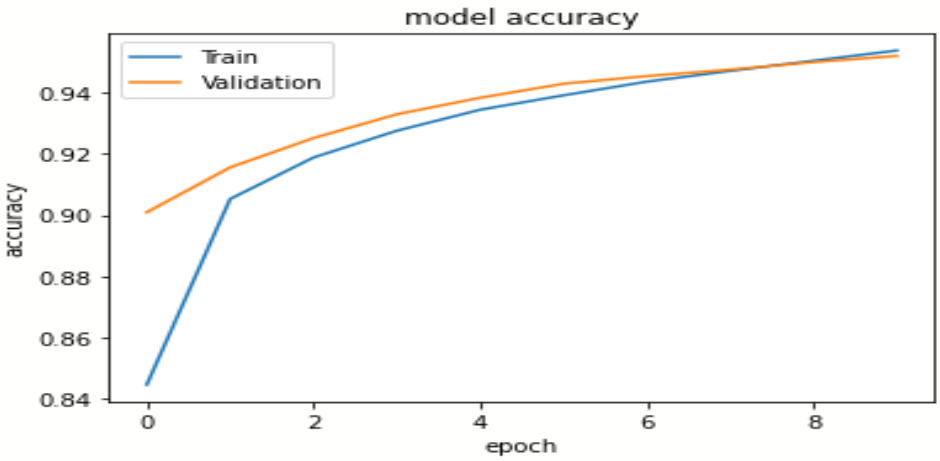
## Making prediction on new data

```
n=random.randint(0,9999)
plt.imshow(x_test[n])
plt.show
```

```
<function matplotlib.pyplot.show(*args, **kw)>
```



## Plot graph for accuracy and loss




## Confusion Matrix

```
test_predict = model.predict(x_test)
# Get the classification labels
test_predict_labels = np.argmax(test_predict, axis=1)
confusion_matrix = tf.math.confusion_matrix(labels=y_test, predictions=test_predict_labels)
print('Confusion matrix of the test set:\n', confusion_matrix)
```

```
Confusion matrix of the test set:
 tf.Tensor(
[[ 965    0    1    0    4    6    1    2    0]
 [   0 1117    2    2    1    1    3    2    7    0]
 [   6    2  977   13    5    1    7    8   12    1]
 [   1    0    9  960    1    9    1   10   13    6]
 [   1    1    4    0  939    1    8    3    3   22]
 [   9    1    2   19    3  823   11    2   14    8]
 [   9    3    2    2   10    7  919    1    5    0]
 [   1   10   18    7    5    1    0  968    1   17]
 [   3    1    3   18    7    7    7    9  916    3]
 [   8    8    0   12   22    3    1   12    6  937]], shape=(10, 10), dtype=int32)
```

## Conclusion

With above code we can see that, throughout the epochs, our model accuracy increases and loss decreases that is good since our model gains confidence with our prediction

This indicates the model is trained in a good way

1. The two loss(loss and val_loss) are decreasing and the accuracy (accuracy and val_accuracy) increasing.
2. The val_accuracy is the measure of how good the model is predicting so, it is observed that the model is well trained after 10 epochs

# References


1. S. Arora and M. P. S. Bhatia, "Handwriting recognition using Deep Learning in Keras," *2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)*, 2018, pp. 142-145, doi: 10.1109/ICACCCN.2018.8748540.
2. https://towardsdatascience.com/feed-forward-neural-networks-how-to-successfully-build-them-in-python-74503409d99a
3. https://pyimagesearch.com/2021/05/06/implementing-feedforward-neural-networks-with-keras-and-tensorflow/
4. https://exchange.scale.com/public/blogs/how-to-build-a-fully-connected-feedforward-neural-network-using-keras-and-tensorflow
5. https://www.kaggle.com/code/prashant111/mnist-deep-neural-network-with-keras/notebook
6. https://sanjayasubedi.com.np/deeplearning/tensorflow-2-first-neural-network-for-fashion-mnist/

Sinhgad Institutes

*Name of the Student:* _____

*Roll no:* _____

*CLASS: - B.E. IT* *Subject Name: - LP-IV Lab*

Assignment No. 03

# ** Build the Image classification model CO2 **

**Date of Performance:** / /2022 **Marks out of 10:**

**Sign with Date:**

# Assignment No. 03

## Problem Statement

Build the Image Classification model by dividing the model into following 4 stages:

7. Import the necessary packages
8. Loading and preprocessing the image data
9. Defining the model's architecture
10. Training the model
11. Estimating the model's performance

## Solution Expected

Implement and train a model for image processing and classification using CNN on a image dataset (Fast food) and improve model generalization by achieving increased accuracy and descresesd loss where model gains good confidence with the prediction.

## Objectives to be achieved

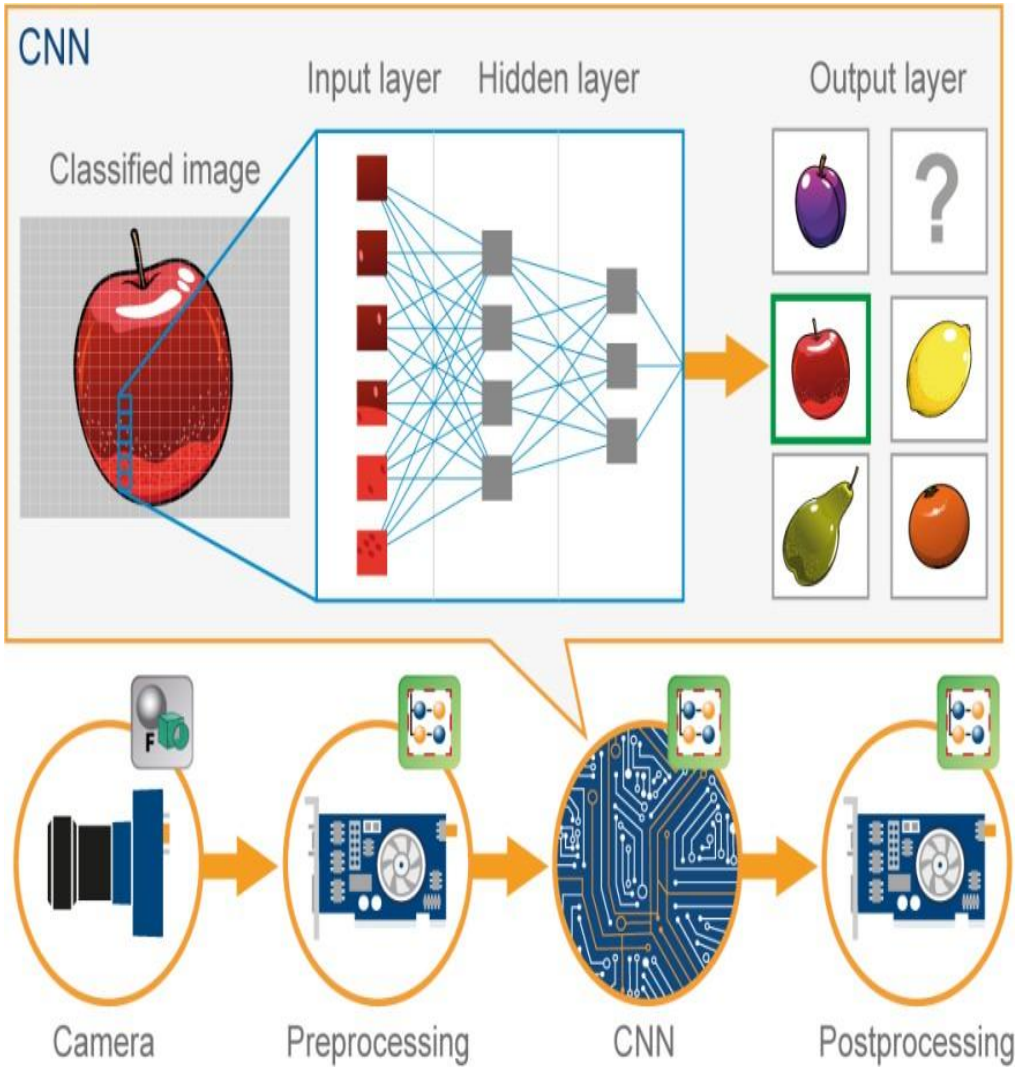
6. Understand how to use Tensorflow Eager and Keras Layers to build neural network architecture.
7. Understand how a model is trained and evaluated.
8. Identify digits from images.
9. Our main goal is to train a neural network (using Keras) to obtain > *90%* accuracy on MNIST dataset.
10. Research at least 1 technique that can be used to improve model generalization.

## Methodology to be used

- Deep Learning
- Convolutional Neural Network

## Introduction

In the past few years, Deep Learning has been proved that it's a very powerful tool due to its ability to handle huge amounts of data. The use of hidden layers exceeds traditional techniques, especially for pattern recognition. One of the most popular Deep Neural Networks is Convolutional Neural Networks (CNN).




A convolutional neural network (CNN) is a type of **Artificial Neural Network (ANN)** used in image recognition and processing which is specially designed for processing data (pixels).

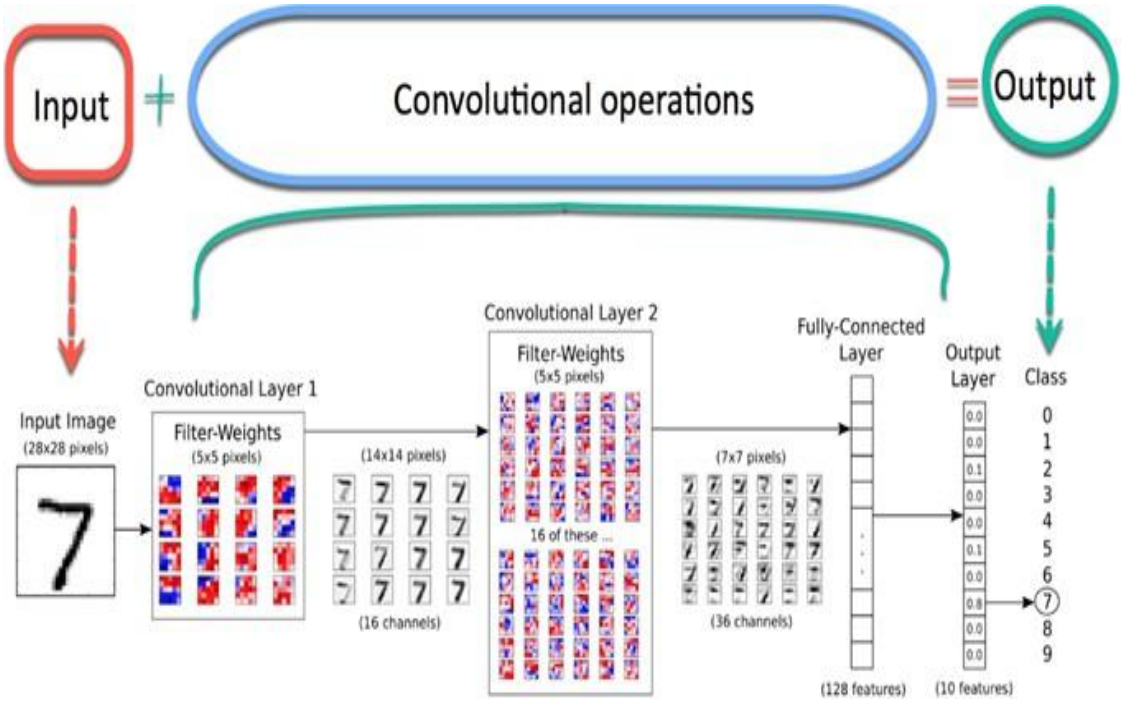



Image Source: Google.com

Before moving further we need to understand what is the neural network? Let's go…

## Neural Network:

A neural network is constructed from several interconnected nodes called **"neurons"**. Neurons are arranged into the **input layer, hidden layer, and output layer.** The input layer corresponds to our predictors/features and the Output layer to our response variable/s.

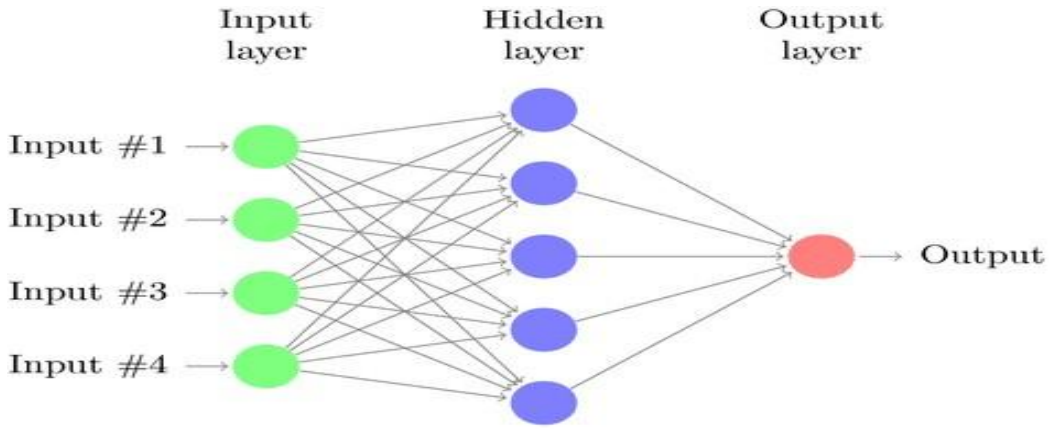


Image Source: Google.com

## Multi-Layer Perceptron (MLP):

The neural network with an input layer, one or more hidden layers, and one output layer is called a **multi-layer perceptron (MLP).** MLP is invented by **Frank Rosenblatt** in the year of 1957. MLP given below has 5 input nodes, 5 hidden nodes with two hidden layers, and one output node

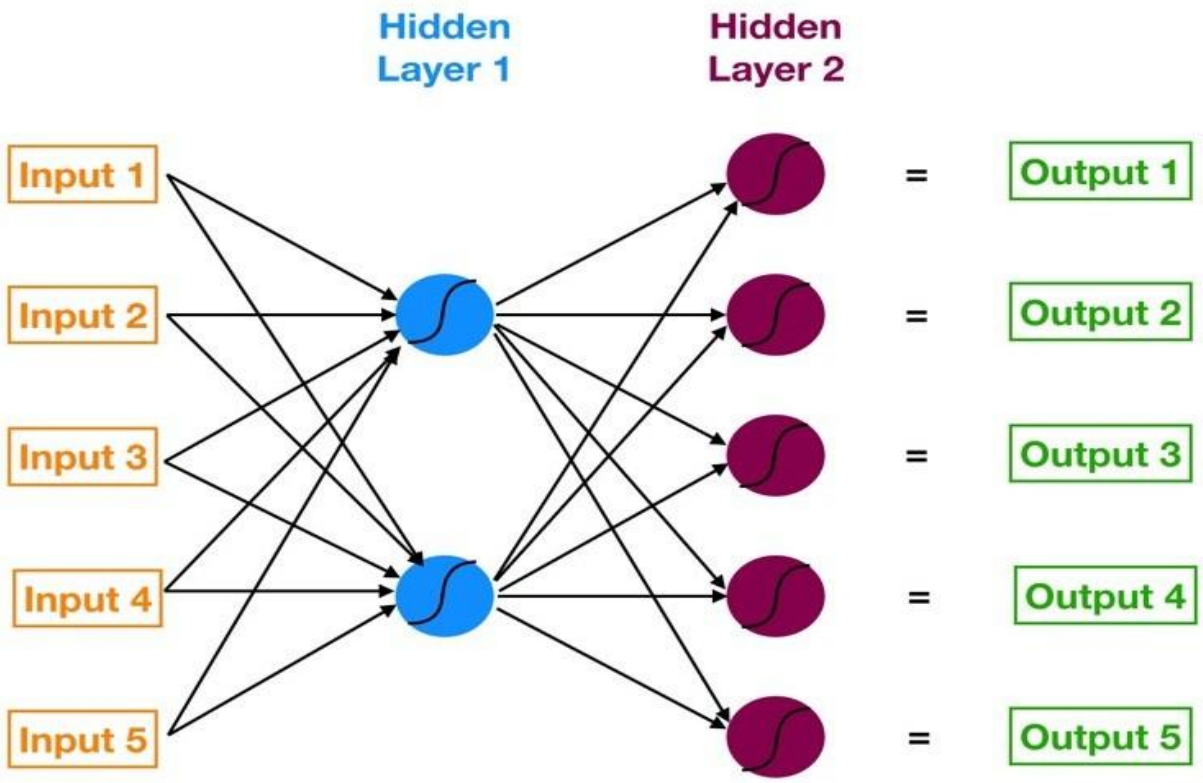



Image Source: Google.com

## How does this Neural Network work?

– Input layer neurons receive incoming information from the data which they process and distribute to the **hidden layers**.

– That information, in turn, is processed by hidden layers and is passed to the output **neurons**.

– The information in this artificial neural network(ANN) is processed in terms of one **activation function**. This function actually imitates the brain neurons.

– Each neuron contains a value of **activation functions** and a **threshold value.**

– The **threshold value** is the minimum value that must be possessed by the input so that it can be activated.

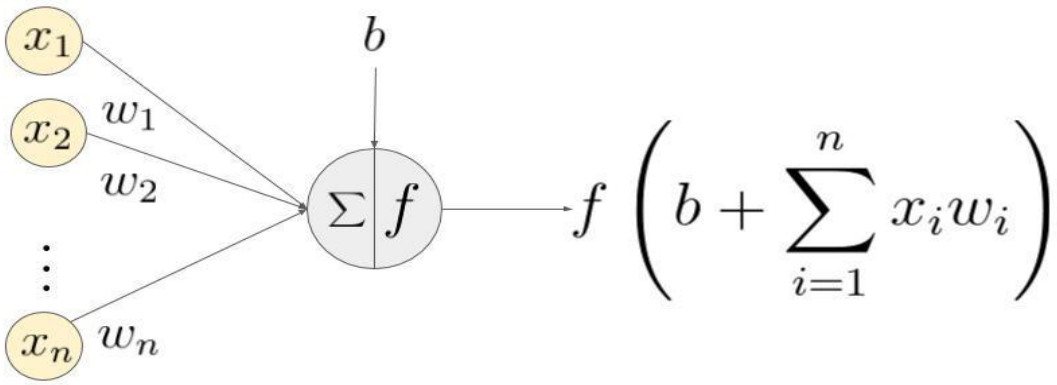
– The task of the neuron is to perform a weighted sum of all the input signals and apply the activation function on the sum before passing it to the next(hidden or output) layer.

## Let us understand what is weightage sum?

Say that, we have values $a1$, $a2$, $a3$, $a4$ for input and weights as $w1$, $w2$, $w3$, $w4$ as the input to one of the hidden layer neuron say $nj$, then the weighted sum is represented as

$$Sj = \sigma\ i{=}1\text{to}4\ wi{*}ai + bj$$

where $bj$ : bias due to node




An example of a neuron showing the input ( $x_1$ - $x_n$ ), their corresponding weights ( $w_1$ - $w_n$ ), a bias ( b ) and the activation function f applied to the weighted sum of the inputs.

Image Source: Google.com

## What are the Activation Functions?

These functions are needed to introduce a non-linearity into the network. The activation function is applied and that output is passed to the next layer.

## *Possible Functions*

- **Sigmoid:** Sigmoid function is differentiable. It produces output between 0 and 1.
- **Hyperbolic Tangent:** Hyperbolic Tangent is also differentiable. This Produces output between -1 and 1.
- **ReLU:** ReLU is most popular function. ReLU is used widely in deep learning.
- **Softmax:** The softmax function is used for multi-class classification problems. It is a generalization of the sigmoid function. It also produces output between 0 and 1

## Convolutional Neural Nework:

Now imagine there is an image of a bird, and you want to identify it whether it is really a bird or something other. The first thing you should do is feed the pixels of the image in the form of arrays to the input layer of the neural network (MLP networks used to classify such things). The hidden layers carry Feature Extraction by performing various calculations and operations. There are multiple hidden layers like the convolution, the ReLU, and the pooling layer that performs feature extraction from your image. So finally, there is a fully connected layer that you can see which identifies the exact object in the image. You can understand very easily from the following figure:

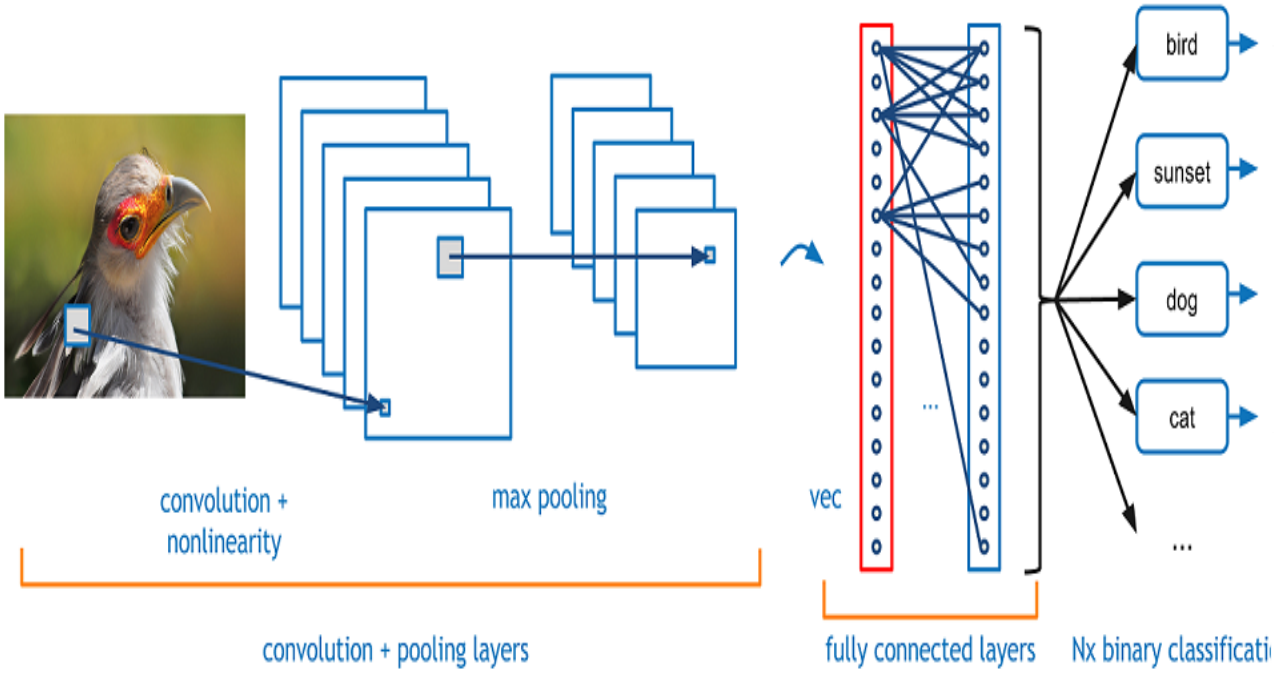



Image Source: Google.com

### *Convolution:-*

Convolution Operation involves matrix arithmetic operations and every image is represented in the form of an array of values (pixels).

Let us understand example:

a = [2, 5, 8, 4, 7, 9]

b = [1, 2, 3]

In Convolution Operation, the arrays are multiplied one by one element-wise, and the product is grouped or summed to create a new array that represents **a*b**.

The first three elements of matrix **a** are now multiplied by the elements of matrix **b**. The product is summed to get the result and stored in a new array of **a*b.**

This process remains continuous until the operation gets completed.

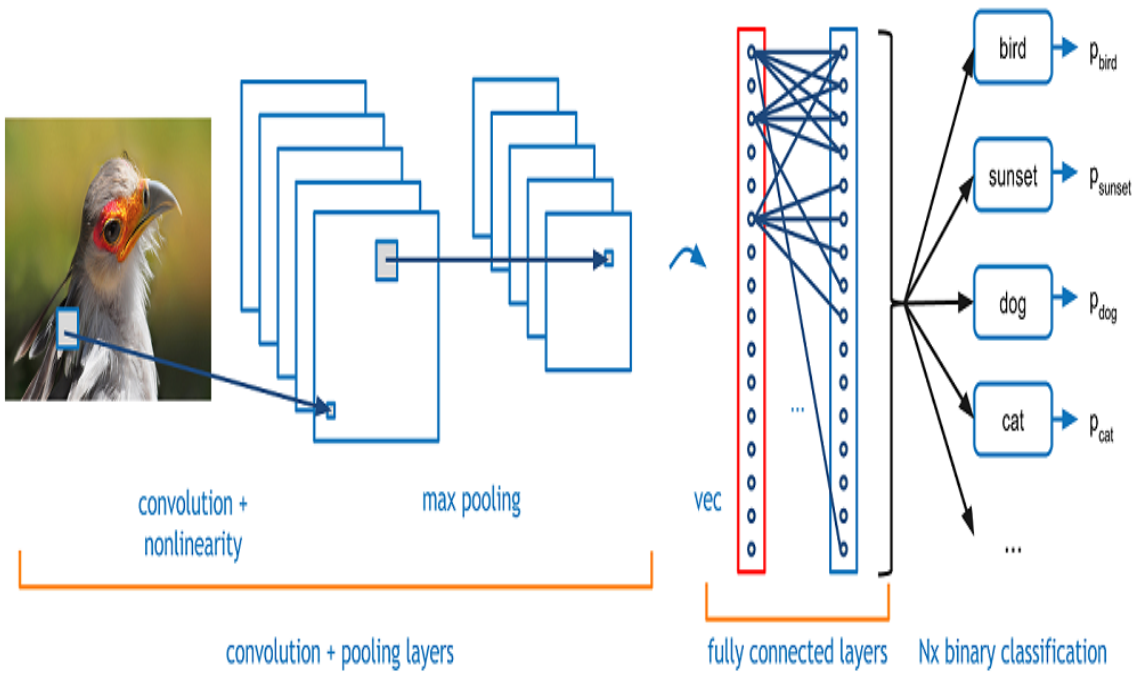



Image Source: Google.com

### *Pooling:*

After the convolution, there is another operation called pooling. So, in the chain, convolution and pooling are applied sequentially on the data in the interest of extracting some features from the data. After the sequential convolutional and pooling layers, the data is flattened into a feed-forward neural network which is also called a Multi-Layer Perceptron.

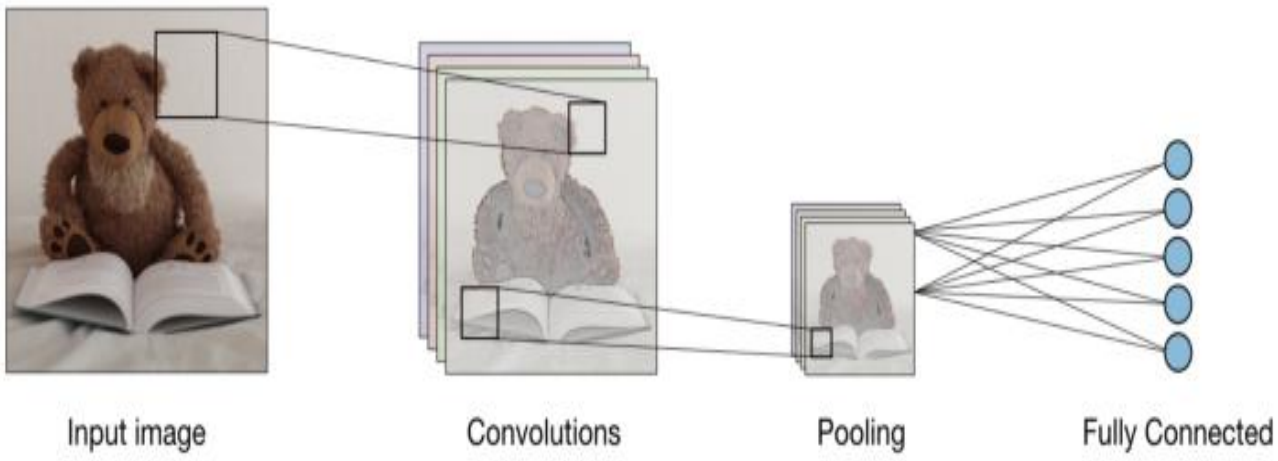


Image Source: Google.com

Up to this point, we have seen concepts that are important for our building CNN model.

***Now we will move forward to see a case study of CNN.***

1) Here we are going to import the necessary libraries which are required for performing CNN tasks.

```
import NumPy as np
%matplotlib inline
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import TensorFlow as tf
tf.compat.v1.set_random_seed (2019)
```


```
import NumPy as np
%matplotlib inline
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import TensorFlow as tf
tf.compat.v1.set_random_seed(2019)
```


2) Here we required the following code to form the CNN model

```
model = tf.keras.models.Sequential([
tf.keras.layers.Conv2D(16,(3,3),activation = "relu" , input_shape = (180,180,3)) ,
```

```
tf.keras.layers.MaxPooling2D(2,2),

tf.keras.layers.Conv2D(32,(3,3),activation = "relu") ,

tf.keras.layers.MaxPooling2D(2,2),

tf.keras.layers.Conv2D(64,(3,3),activation = "relu") ,

tf.keras.layers.MaxPooling2D(2,2),

tf.keras.layers.Conv2D(128,(3,3),activation = "relu"),

tf.keras.layers.MaxPooling2D(2,2),

tf.keras.layers.Flatten(),

tf.keras.layers.Dense(550,activation="relu"),      #Adding the Hidden layer

tf.keras.layers.Dropout(0.1,seed = 2019),

tf.keras.layers.Dense(400,activation ="relu"),

tf.keras.layers.Dropout(0.3,seed = 2019),

tf.keras.layers.Dense(300,activation="relu"),

tf.keras.layers.Dropout(0.4,seed = 2019),

tf.keras.layers.Dense(200,activation ="relu"),

tf.keras.layers.Dropout(0.2,seed = 2019),

tf.keras.layers.Dense(5,activation = "softmax")   #Adding the Output Layer])
```

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(16,(3,3),activation = "relu" , input_shape = (18
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(32,(3,3),activation = "relu") ,
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(64,(3,3),activation = "relu") ,
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(128,(3,3),activation = "relu"),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(550,activation="relu"),      #Adding the Hidden I
    tf.keras.layers.Dropout(0.1,seed = 2019),
    tf.keras.layers.Dense(400,activation ="relu"),
    tf.keras.layers.Dropout(0.3,seed = 2019),
    tf.keras.layers.Dense(300,activation="relu"),
    tf.keras.layers.Dropout(0.4,seed = 2019),
    tf.keras.layers.Dense(200,activation ="relu"),
    tf.keras.layers.Dropout(0.2,seed = 2019),
    tf.keras.layers.Dense(5,activation = "softmax")   #Adding the Output La
])
```

A convoluted image can be too large and so it is reduced without losing features or patterns, so pooling is done.

Here Creating a Neural network is to initialize the network using the Sequential model from Keras.

Flatten () - Flattening transforms a two-dimensional matrix of features into a vector of features.

3) Now let's see a summary of the CNN model

model.summary ()

```
model.summary()
```

It will print the following output

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 178, 178, 16)      448
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 89, 89, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 87, 87, 32)        4640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 43, 43, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 41, 41, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 20, 20, 64)        0
```

It will print the following output

Model: "sequential"

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d (Conv2D) | (None, 178, 178, 16) | 448 |
| max_pooling2d (MaxPooling2D) | (None, 89, 89, 16) | 0 |
| conv2d_1 (Conv2D) | (None, 87, 87, 32) | 4640 |
| max_pooling2d_1 (MaxPooling2 | (None, 43, 43, 32) | 0 |
| conv2d_2 (Conv2D) | (None, 41, 41, 64) | 18496 |
| max_pooling2d_2 (MaxPooling2 | (None, 20, 20, 64) | 0 |
| conv2d_3 (Conv2D) | (None, 18, 18, 128) | 73856 |
| max_pooling2d_3 (MaxPooling2 | (None, 9, 9, 128) | 0 |
| flatten (Flatten) | (None, 10368) | 0 |

| | | |
|---|---|---|
| dense (Dense) | (None, 550) | 5702950 |
| dropout (Dropout) | (None, 550) | 0 |
| dense_1 (Dense) | (None, 400) | 220400 |
| dropout_1 (Dropout) | (None, 400) | 0 |
| dense_2 (Dense) | (None, 300) | 120300 |
| dropout_2 (Dropout) | (None, 300) | 0 |
| dense_3 (Dense) | (None, 200) | 60200 |
| dropout_3 (Dropout) | (None, 200) | 0 |
| dense_4 (Dense) | (None, 5) | 1005 |

==========
=

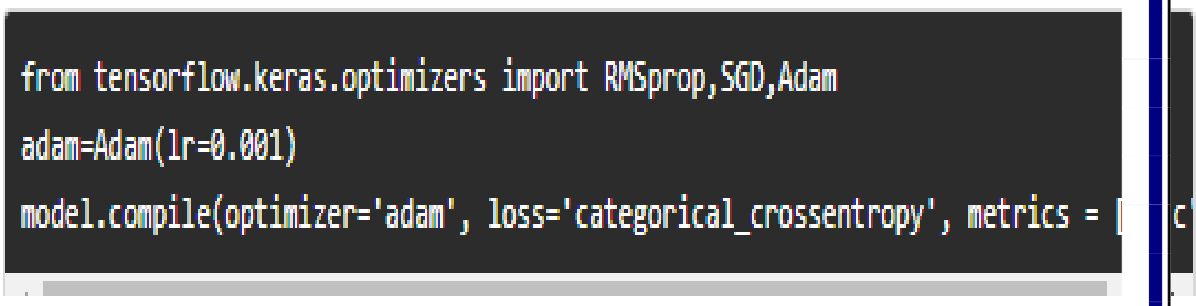
Total params: 6,202,295

Trainable params: 6,202,295

Non-trainable params: 0

4) So now we are required to specify optimizers.

```
from tensorflow.keras.optimizers import RMSprop,SGD,Adam

adam=Adam(lr=0.001)

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics = ['acc'])
```



- Optimizer is used to reduce the cost calculated by cross-entropy
- The loss function is used to calculate the error
- The metrics term is used to represent the efficiency of the model

5) In this step, we will see how to set the data directory and generate image data.

```
bs=30        #Setting batch size

train_dir = "D:/Data Science/Image Datasets/FastFood/train/"    #Setting training directory

validation_dir = "D:/Data Science/Image Datasets/FastFood/test/"    #Setting testing directory

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# All images will be rescaled by 1./255.

train_datagen = ImageDataGenerator( rescale = 1.0/255. )

test_datagen  = ImageDataGenerator( rescale = 1.0/255. )

# Flow training images in batches of 20 using train_datagen generator

#Flow_from_directory function lets the classifier directly identify the labels from the name of the directories the image lies in

train_generator=train_datagen.flow_from_directory(train_dir,batch_size=bs,class_mode='categorical',target_size=(180,180))

# Flow validation images in batches of 20 using test_datagen generator

validation_generator = test_datagen.flow_from_directory(validation_dir,  batch_size=bs,

                                    class_mode  = 'categorical', target_size=(180,180))
```

```
bs=30         #Setting batch size
train_dir = "D:/Data Science/Image Datasets/FastFood/train/"   #Setting training
validation_dir = "D:/Data Science/Image Datasets/FastFood/test/"   #Setting testi
from tensorflow.keras.preprocessing.image import ImageDataGenerator
# All images will be rescaled by 1./255.
train_datagen = ImageDataGenerator( rescale = 1.0/255. )
test_datagen  = ImageDataGenerator( rescale = 1.0/255. )
# Flow training images in batches of 20 using train_datagen generator
#Flow_from_directory function lets the classifier directly identify the labels fr
train_generator=train_datagen.flow_from_directory(train_dir,batch_size=bs,class_m
# Flow validation images in batches of 20 using test_datagen generator
validation_generator =  test_datagen.flow_from_directory(validation_dir,
                                                         batch_size=bs,
                                                         class_mode  = 'categoric
                                                         target_size=(180,180))
```

The output will be:

- Found 1465 images belonging to 5 classes.
- Found 893 images belonging to 5 classes.

6) Final step of the fitting model.

```
history = model.fit(train_generator,
                    validation_data=validation_generator,
                    steps_per_epoch=150 // bs,
                    epochs=30,
```

validation_steps=50 // bs,

verbose=2)

The output will be:

```
Found 1465 images belonging to 5 classes.
Found 893 images belonging to 5 classes.
```

6) Final step of the fitting model.

```
history = model.fit(train_generator,
                    validation_data=validation_generator,
                    steps_per_epoch=150 // bs,
                    epochs=30,
                    validation_steps=50 // bs,
                    verbose=2)
```

The output will be:

Epoch 1/30

5/5 - 4s - loss: 0.8625 - acc: 0.6933 - val_loss: 1.1741 - val_acc: 0.5000

Epoch 2/30

5/5 - 3s - loss: 0.7539 - acc: 0.7467 - val_loss: 1.2036 - val_acc: 0.5333

Epoch 3/30

5/5 - 3s - loss: 0.7829 - acc: 0.7400 - val_loss: 1.2483 - val_acc: 0.5667

Epoch 4/30

5/5 - 3s - loss: 0.6823 - acc: 0.7867 - val_loss: 1.3290 - val_acc: 0.4333

Epoch 5/30

5/5 - 3s - loss: 0.6892 - acc: 0.7800 - val_loss: 1.6482 - val_acc: 0.4333

Epoch 6/30

5/5 - 3s - loss: 0.7903 - acc: 0.7467 - val_loss: 1.0440 - val_acc: 0.6333

Epoch 7/30

5/5 - 3s - loss: 0.5731 - acc: 0.8267 - val_loss: 1.5226 - val_acc: 0.5000

Epoch 8/30

5/5 - 3s - loss: 0.5949 - acc: 0.8333 - val_loss: 0.9984 - val_acc: 0.6667

Epoch 9/30

5/5 - 3s - loss: 0.6162 - acc: 0.8069 - val_loss: 1.1490 - val_acc: 0.5667

Epoch 10/30

5/5 - 3s - loss: 0.7509 - acc: 0.7600 - val_loss: 1.3168 - val_acc: 0.5000

Epoch 11/30

5/5 - 4s - loss: 0.6180 - acc: 0.7862 - val_loss: 1.1918 - val_acc: 0.7000

Epoch 12/30

5/5 - 3s - loss: 0.4936 - acc: 0.8467 - val_loss: 1.0488 - val_acc: 0.6333

Epoch 13/30

5/5 - 3s - loss: 0.4290 - acc: 0.8400 - val_loss: 0.9400 - val_acc: 0.6667

Epoch 14/30

5/5 - 3s - loss: 0.4205 - acc: 0.8533 - val_loss: 1.0716 - val_acc: 0.7000

Epoch 15/30

5/5 - 4s - loss: 0.5750 - acc: 0.8067 - val_loss: 1.2055 - val_acc: 0.6000

Epoch 16/30

5/5 - 4s - loss: 0.4080 - acc: 0.8533 - val_loss: 1.5014 - val_acc: 0.6667

Epoch 17/30

5/5 - 3s - loss: 0.3686 - acc: 0.8467 - val_loss: 1.0441 - val_acc: 0.5667

Epoch 18/30

5/5 - 3s - loss: 0.5474 - acc: 0.8067 - val_loss: 0.9662 - val_acc: 0.7333

Epoch 19/30

5/5 - 3s - loss: 0.5646 - acc: 0.8138 - val_loss: 0.9151 - val_acc: 0.7000

Epoch 20/30

5/5 - 4s - loss: 0.3579 - acc: 0.8800 - val_loss: 1.4184 - val_acc: 0.5667

Epoch 21/30

5/5 - 3s - loss: 0.3714 - acc: 0.8800 - val_loss: 2.0762 - val_acc: 0.6333

Epoch 22/30

5/5 - 3s - loss: 0.3654 - acc: 0.8933 - val_loss: 1.8273 - val_acc: 0.5667

Epoch 23/30

5/5 - 3s - loss: 0.3845 - acc: 0.8933 - val_loss: 1.0199 - val_acc: 0.7333

Epoch 24/30

5/5 - 3s - loss: 0.3356 - acc: 0.9000 - val_loss: 0.5168 - val_acc: 0.8333

Epoch 25/30

5/5 - 3s - loss: 0.3612 - acc: 0.8667 - val_loss: 1.7924 - val_acc: 0.5667

Epoch 26/30

5/5 - 3s - loss: 0.3075 - acc: 0.8867 - val_loss: 1.0720 - val_acc: 0.6667

Epoch 27/30

5/5 - 3s - loss: 0.2820 - acc: 0.9400 - val_loss: 2.2798 - val_acc: 0.5667

Epoch 28/30

5/5 - 3s - loss: 0.3606 - acc: 0.8621 - val_loss: 1.2423 - val_acc: 0.8000

Epoch 29/30

5/5 - 3s - loss: 0.2630 - acc: 0.9000 - val_loss: 1.4235 - val_acc: 0.6333

Epoch 30/30

5/5 - 3s - loss: 0.3790 - acc: 0.9000 - val_loss: 0.6173 - val_acc: 0.8000

The above function trains the neural network using the training set and evaluates its performance on the test set. The functions return two metrics for each epoch ‘acc’ and ‘val_acc’ which are the accuracy of predictions obtained in the training set and accuracy attained in the test set respectively.

**Conclusion:**

Thus we have created a machine learning model for image processing and classification using CNN and found sufficient accuracy.

Sinhgad Institutes

*Name of the Student:* _____

*Roll no:* _____

*CLASS: - B. E. IT* *Subject Name: - LP-IV Lab*

Assignment No. 04

# ** Use Autoencoder to implement anomaly detection.: CO2**

**Date of Performance:** / /2022

**Marks out of 10:**

**Sign with Date:**

Sinhgad Institutes

*Name of the Student:* _____

*Roll no:* _____

*CLASS: - B..E. IT* *Subject Name: - LP-IV Lab*

Assignment No. 05

# ** Implement the Continuous Bag of Words Model: CO3 **

**Date of Performance:** / /2022 **Marks out of 10:**

**Sign with Date:**

**Assignment No: 05**

Sinhgad Institutes

*Name of the Student:* _____

*Roll no:* _____

*CLASS: - B..E. IT* *Subject Name: - LP-IV Lab*

Assignment No. 06

**** Object detection using Transfer Learning of CNN archit.:CO3 ****

**Date of Performance:** / /2022 **Marks out of 10:**

**Sign with Date:**

## Assignment No : 06

# Assignment No. 04

## Problem Statement

*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

Solution Expected
AutoEncoders are widely used in anomaly detection. The reconstruction errors are used as the anomaly scores. Let us look at how we can use AutoEncoder for anomaly detection using TensorFlow.

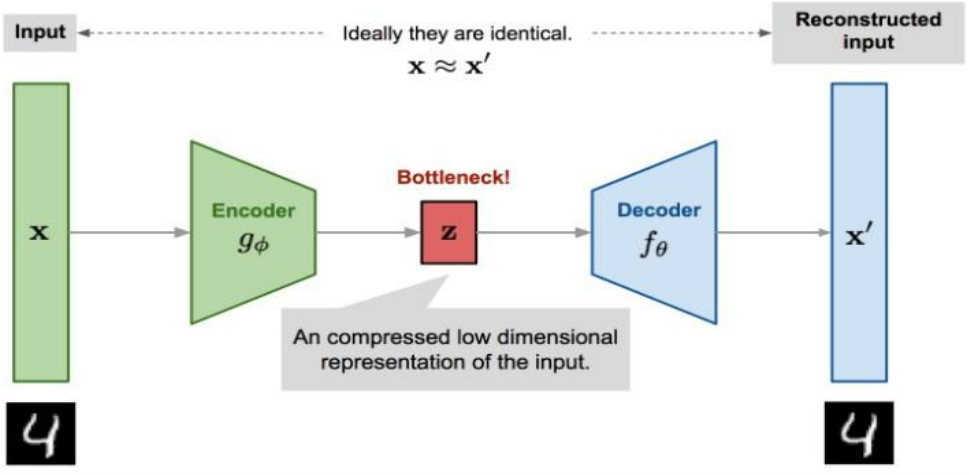
Import the required libraries and load the data. Here we are using the ECG data which consists of labels 0 and 1. Label 0 denotes the observation as an anomaly and label 1 denotes the observation as normal.

Objectives to be achieved

1)Use Autoencoder to implement anomaly detection.

Methodology to be used

AutoEncoder is a generative unsupervised deep learning algorithm used for reconstructing high-dimensional input data using a neural network with a narrow bottleneck layer in the middle which containsthe latent representation of the input data.



## Import required libraries

```
import pandas as pd
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split


from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, recall_score,
accuracy_score, precision_score


RANDOM_SEED = 2021
TEST_PCT = 0.3
LABELS = ["Normal","Fraud"]
```

## Read the dataset

I had downloaded the data from Kaggle and stored it in the local directory.

```
dataset = pd.read_csv("creditcard.csv")
```

## Exploratory Data Analysis

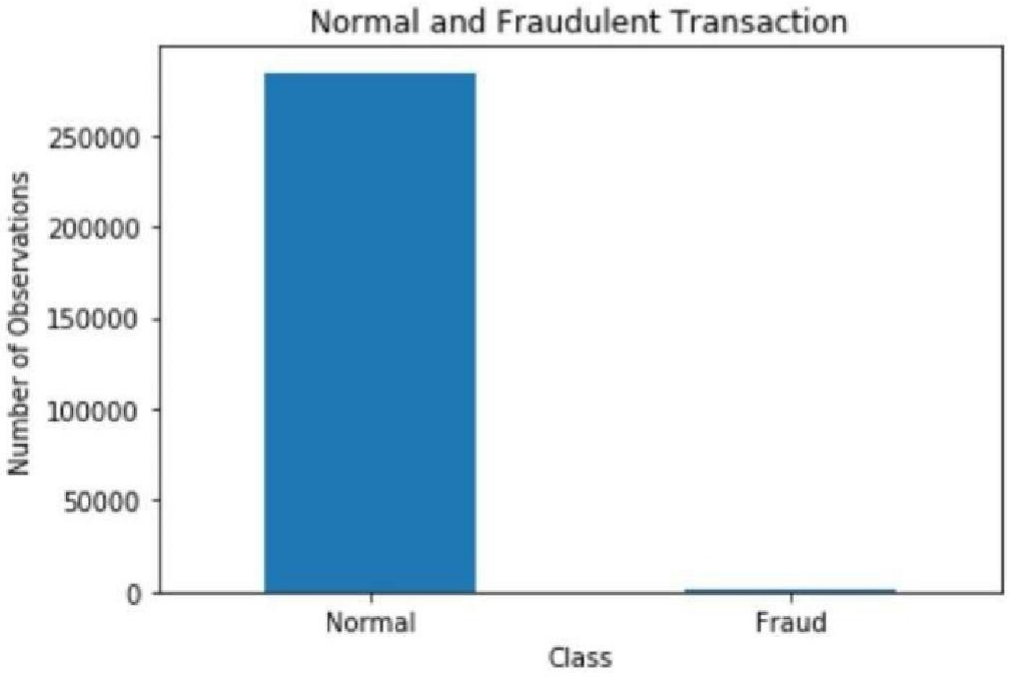
```
#check for any  nullvalues
print("Any nulls in the dataset
",dataset.isnull().values.any() )
print('-------')
print("No. of unique labels ",
len(dataset['Class'].unique()))
print("Label values
",dataset.Class.unique())

#0 is for normal credit card transaction
#1 is for fraudulent credit card
transaction
print('-------')
print("Break down of the Normal and Fraud
Transactions")
print(pd.value_counts(dataset['Class'],
sort = True) )
```

```
Any nulls in the dataset  False
-------
No. of unique labels  2
Label values  [0 1]
-------
Break down of the Normal and Fraud Transactions
0    284315
1       492
Name: Class, dtype: int64
```

## Visualize the dataset

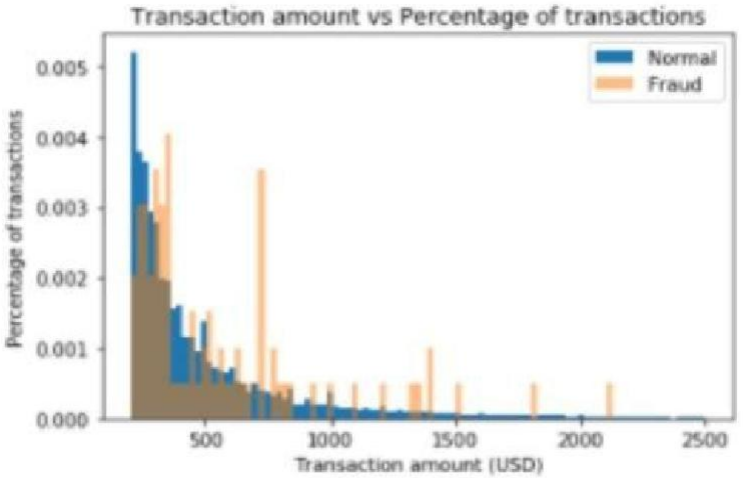
plotting the number of normal and fraud transactions in the dataset.

```
#Visualizing the imbalanced dataset
count_classes =
pd.value_counts(dataset['Class'], sort =
True)
count_classes.plot(kind = 'bar', rot=0)
plt.xticks(range(len(dataset['Class'].uni
que())), dataset.Class.unique())
plt.title("Frequency by observation
number")
plt.xlabel("Class")
plt.ylabel("Number of Observations");
```



Visualizing the amount for normal and fraud transactions.

```
# Save the normal and fradulent
transactions in separate dataframe
normal_dataset = dataset[dataset.Class ==
0]
fraud_dataset = dataset[dataset.Class ==
1]

#Visualize transactionamounts for normal
and fraudulent transactions
bins = np.linspace(200, 2500, 100)
plt.hist(normal_dataset.Amount,
bins=bins, alpha=1, density=True,
label='Normal')
plt.hist(fraud_dataset.Amount, bins=bins,
alpha=0.5, density=True, label='Fraud')
plt.legend(loc='upper right')
plt.title("Transaction amount vs
Percentage of transactions")
plt.xlabel("Transaction amount (USD)")
plt.ylabel("Percentage of transactions");
plt.show()
```



## Create train and test dataset

## Checking on the dataset

| V6 | V7 | V8 | V9 | V10 | ... | V22 | V23 | V24 | V25 | V26 | V27 | V28 | Time | Amount | Class |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0.462388 | 0.239599 | 0.098698 | 0.363787 | 0.090794 | ... | 0.277838 | -0.110474 | 0.066928 | 0.128539 | -0.189115 | 0.133558 | -0.021053 | 0.0 | 149.62 | 0 |
| -0.082361 | -0.078803 | 0.085102 | -0.255425 | -0.166974 | ... | -0.638672 | 0.101288 | -0.339846 | 0.167170 | 0.125895 | -0.008983 | 0.014724 | 0.0 | 2.69 | 0 |
| 1.800499 | 0.791461 | 0.247676 | -1.514654 | 0.207643 | ... | 0.771679 | 0.909412 | -0.689281 | -0.327642 | -0.139097 | -0.055353 | -0.059752 | 1.0 | 378.66 | 0 |
| 1.247203 | 0.237609 | 0.377436 | -1.387024 | -0.054952 | ... | 0.005274 | -0.190321 | -1.175575 | 0.647376 | -0.221929 | 0.062723 | 0.061458 | 1.0 | 123.50 | 0 |
| 0.095921 | 0.592941 | -0.270533 | 0.817739 | 0.753074 | ... | 0.798278 | -0.137458 | 0.141267 | -0.206010 | 0.502292 | 0.219422 | 0.215153 | 2.0 | 69.99 | 0 |

Time and Amount are the columns that are not scaled, so applying StandardScaler to only Amount and Time columns. Normalizing thevalues between 0 and 1 did not work great for the dataset.

```
sc=StandardScaler()
dataset['Time'] =
sc.fit_transform(dataset['Time'].values.r
eshape(-1, 1))
dataset['Amount'] =
sc.fit_transform(dataset['Amount'].values
.reshape(-1, 1))
```

The last column in the dataset is our target variable.

The last column in the dataset is our target variable.

```
raw_data = dataset.values
# The last element contains if the
transaction is normal which is
represented by a 0 and if fraud then 1
labels = raw_data[:, -1]

# The other data points are the
electrocadriogram data
data = raw_data[:, 0:-1]

train_data, test_data, train_labels,
test_labels = train_test_split(
    data, labels, test_size=0.2,
random_state=2021
)
```

**Normalise the data to have a value between 0 and 1**

## Normalize the data to have a value between 0 and 1

```
min_val = tf.reduce_min(train_data)
max_val = tf.reduce_max(train_data)

train_data = (train_data - min_val) /
(max_val - min_val)
test_data = (test_data - min_val) /
(max_val - min_val)

train_data = tf.cast(train_data,
tf.float32)
test_data = tf.cast(test_data,
tf.float32)
```

***Use only normal transactions to train the Autoencoder.***

Normal data has a value of 0 in the target variable. Using the target variable to create a normal and fraud dataset.

```
train_labels = train_labels.astype(bool)
test_labels = test_labels.astype(bool)

#creating normal and fraud datasets
normal_train_data =
train_data[~train_labels]
normal_test_data =
test_data[~test_labels]

fraud_train_data =
train_data[train_labels]
fraud_test_data = test_data[test_labels]
print(" No. of records in Fraud Train
Data=",len(fraud_train_data))
print(" No. of records in Normal Train
data=",len(normal_train_data))
print(" No. of records in Fraud Test
Data=",len(fraud_test_data))
print(" No. of records in Normal Test
data=",len(normal_test_data))
```

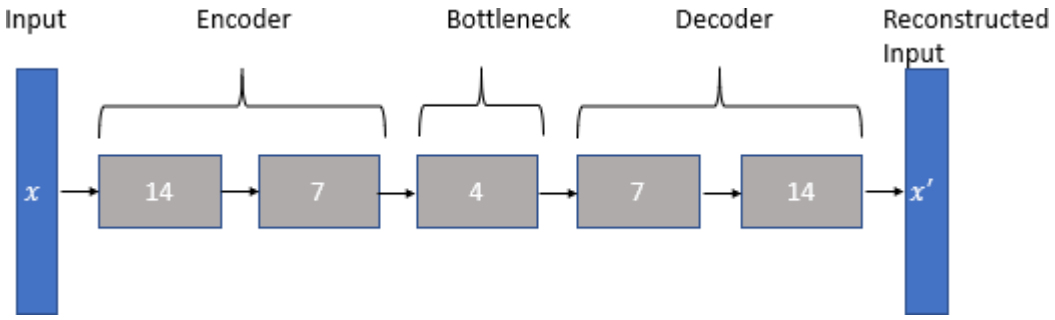
**Set the training parameter values**

Set the training parameter values

```
nb_epoch = 50
batch_size = 64
input_dim = normal_train_data.shape[1]
#num of columns, 30
encoding_dim = 14
hidden_dim_1 = int(encoding_dim / 2) #
hidden_dim_2=4
learning_rate = 1e-7
```

**Create the Autoencoder**

The architecture of the autoencoder is shown below.



```
#input Layer
input_layer =
tf.keras.layers.Input(shape=(input_dim,
))

#Encoder

encoder =
tf.keras.layers.Dense(encoding_dim,
activation="tanh",
activity_regularizer=tf.keras.regularizer
s.l2(learning_rate))(input_layer)
encoder=tf.keras.layers.Dropout(0.2)
(encoder)
encoder =
tf.keras.layers.Dense(hidden_dim_1,
activation='relu')(encoder)
encoder =
tf.keras.layers.Dense(hidden_dim_2,
activation=tf.nn.leaky_relu)(encoder)

# Decoder
decoder =
tf.keras.layers.Dense(hidden_dim_1,
activation='relu')(encoder)
decoder=tf.keras.layers.Dropout(0.2)
(decoder)
decoder =
tf.keras.layers.Dense(encoding_dim,
activation='relu')(decoder)
decoder =
tf.keras.layers.Dense(input_dim,
activation='tanh')(decoder)

#Autoencoder
autoencoder =
tf.keras.Model(inputs=input_layer,
outputs=decoder)
autoencoder.summary()
```

```
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 30)]              0
_________________________________________________________________
dense (Dense)                (None, 14)                434
_________________________________________________________________
dropout (Dropout)            (None, 14)                0
_________________________________________________________________
dense_1 (Dense)              (None, 7)                 105
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 32
_________________________________________________________________
dense_3 (Dense)              (None, 7)                 35
_________________________________________________________________
dropout_1 (Dropout)          (None, 7)                 0
_________________________________________________________________
dense_4 (Dense)              (None, 14)                112
_________________________________________________________________
dense_5 (Dense)              (None, 30)                450
=================================================================
Total params: 1,168
Trainable params: 1,168
Non-trainable params: 0
```

### Define the callbacks for checkpoints and early stopping

```
cp =
tf.keras.callbacks.ModelCheckpoint(filepa
th="autoencoder_fraud.h5",

mode='min', monitor='val_loss',
verbose=2, save_best_only=True)
# define our early stopping
early_stop =
tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    min_delta=0.0001,
    patience=10,
    verbose=1,
    mode='min',
    restore_best_weights=True
```

**Plot training and test loss**

## Compile the Autoencoder

```
autoencoder.compile(metrics=['accuracy'],

loss='mean_squared_error',
                    optimizer='adam')
```

## Train the Autoencoder

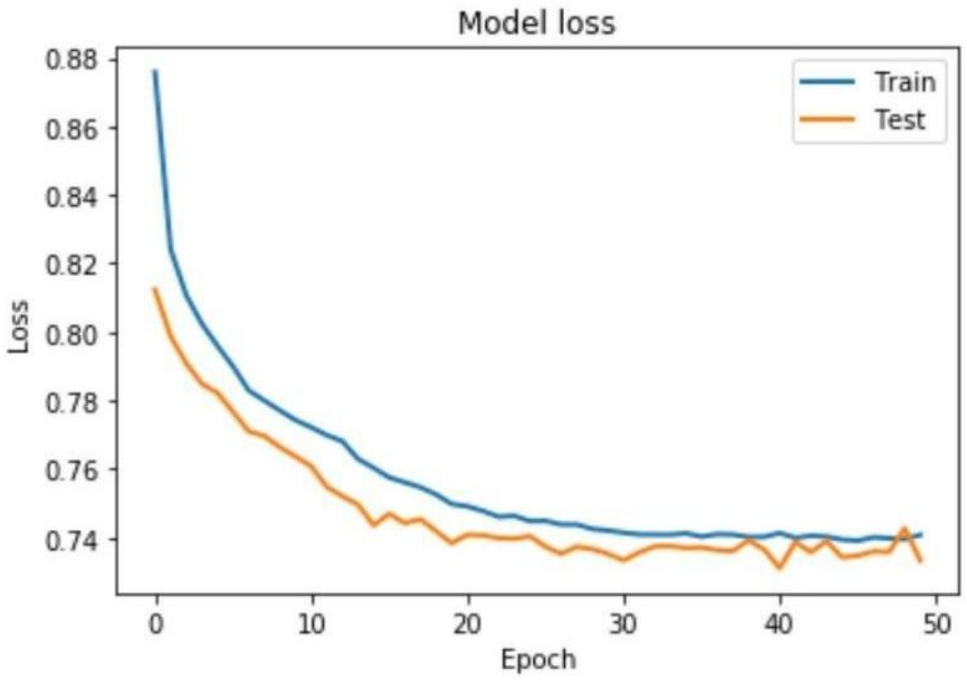
```
history =
autoencoder.fit(normal_train_data,
normal_train_data,
                    epochs=nb_epoch,

batch_size=batch_size,
                    shuffle=True,
                    validation_data=
(test_data, test_data),
                    verbose=1,
                    callbacks=[cp,
early_stop]
                    ).history
```

```
Epoch 11/25
7085/7108 [============================>.] - ETA: 0s - loss: 5.3796e-04
Epoch 00011: val_loss did not improve from 0.00053
Restoring model weights from the end of the best epoch.
7108/7108 [==============================] - 8s 1ms/step - loss: 5.3799e-04 - val_loss: 5.3531e-04
Epoch 00011: early stopping
```

**Plot training and test loss**

## Plot training and test loss

```
plt.plot(history['loss'], linewidth=2,
label='Train')
plt.plot(history['val_loss'],
linewidth=2, label='Test')
plt.legend(loc='upper right')
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
#plt.ylim(ymin=0.70,ymax=1)
plt.show()
```




**Detect Anomalies on test data**

***Anomalies are data points where the reconstruction lossis higher***

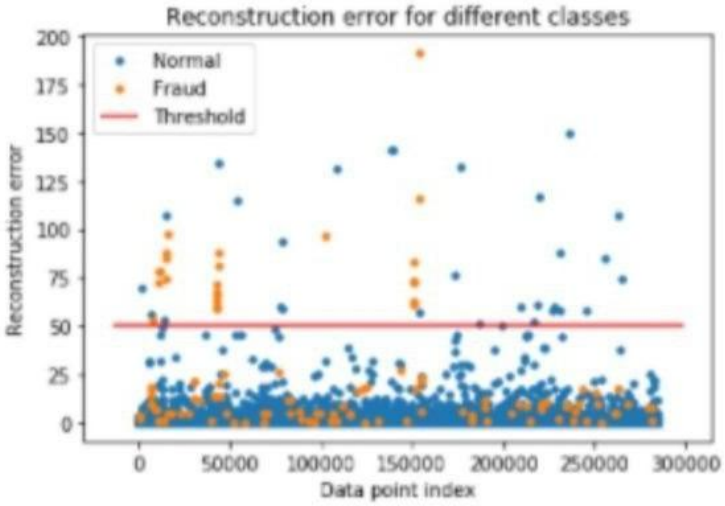
To calculate the reconstruction loss on test data, predict the test data and calculate the mean square error between the test data andthe reconstructed test data.

```
test_x_predictions =
autoencoder.predict(test_data)
mse = np.mean(np.power(test_data -
test_x_predictions, 2), axis=1)
error_df =
pd.DataFrame({'Reconstruction_error':
mse,
                        'True_class':
test_labels})
```

Plotting the test data points and their respective reconstruction

error sets a threshold value to visualize if the threshold value needsto be adjusted.

```
threshold_fixed = 50
groups = error_df.groupby('True_class')
fig, ax = plt.subplots()

for name, group in groups:
    ax.plot(group.index,
group.Reconstruction_error, marker='o',
ms=3.5, linestyle='',
            label= "Fraud" if name == 1
else "Normal")
ax.hlines(threshold_fixed, ax.get_xlim()
[0], ax.get_xlim()[1], colors="r",
zorder=100, label='Threshold')
ax.legend()
plt.title("Reconstruction error for
normal and fraud data")
plt.ylabel("Reconstruction error")
plt.xlabel("Data point index")
plt.show();
```



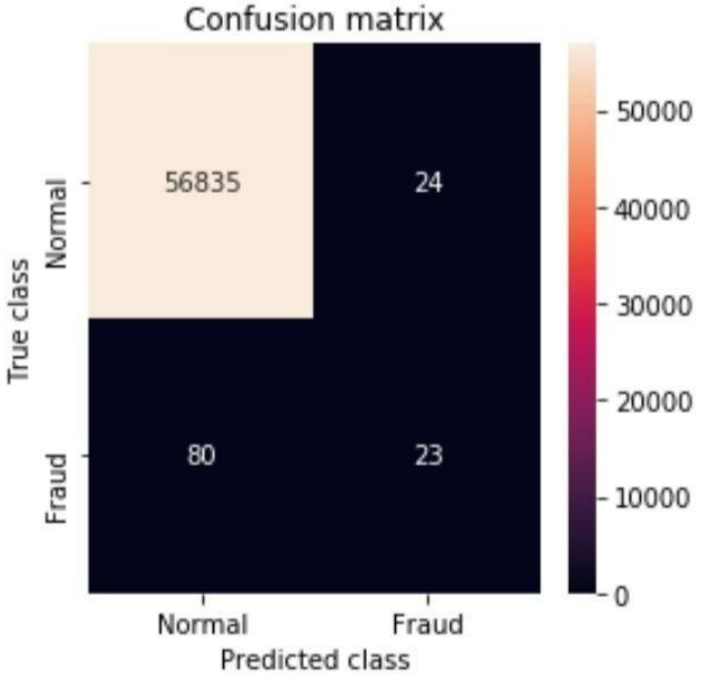
Detect anomalies as points where the reconstruction loss is greaterthan a fixed threshold. Here we see that a value of 52 for the threshold will be good.

## Evaluating the performance of the anomaly detection

```
threshold_fixed =52
pred_y = [1 if e > threshold_fixed else 0
for e in
error_df.Reconstruction_error.values]
error_df['pred'] =pred_y
conf_matrix =
confusion_matrix(error_df.True_class,
pred_y)

plt.figure(figsize=(4, 4))
sns.heatmap(conf_matrix,
xticklabels=LABELS, yticklabels=LABELS,
annot=True, fmt="d");
plt.title("Confusion matrix")
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.show()

# print Accuracy, precision and recall
print(" Accuracy:
",accuracy_score(error_df['True_class'],
error_df['pred']))
print(" Recall:
",recall_score(error_df['True_class'],
error_df['pred']))
print(" Precision:
",precision_score(error_df['True_class'],
error_df['pred']))
```

## Evaluating the performance of the anomaly detection

```
threshold_fixed =52
pred_y = [1 if e > threshold_fixed else 0
for e in
error_df.Reconstruction_error.values]
error_df['pred'] =pred_y
conf_matrix =
confusion_matrix(error_df.True_class,
pred_y)

plt.figure(figsize=(4, 4))
sns.heatmap(conf_matrix,
xticklabels=LABELS, yticklabels=LABELS,
annot=True, fmt="d");
plt.title("Confusion matrix")
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.show()

# print Accuracy, precision and recall
print(" Accuracy:
",accuracy_score(error_df['True_class'],
error_df['pred']))
print(" Recall:
",recall_score(error_df['True_class'],
error_df['pred']))
print(" Precision:
",precision_score(error_df['True_class'],
error_df['pred']))
```




```
Accuracy:  0.9981742214107651
Recall:  0.22330097087378642
Precision:  0.48936170212765956
```

As our dataset is highly imbalanced, we see a high accuracy but a low recall and precision.

Things to further improve precision and recall would add more relevant features, different architecture for autoencoder, different hyperparameters, or a different algorithm.

## Conclusion:

Autoencoders can be used as an anomaly detection algorithm when we have an unbalanced dataset where we have a lot of good examples and only a few anomalies. Autoencoders are trained to minimise reconstruction error. When we train the autoencoders on normal data or good data, we can hypothesise that the anomalies will have higher reconstruction errors than the good or normal data.

# 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

**Theory :**

**1)**What is **NLP** ?
2) What is Word embedding related to NLP ?
3) Explain Word2Vec techniques.
4) Enlist applications of Word embedding in NLP.
5) Explain CBOW architecture.
6) What will be input to CBOW model and Output to CBW model.
**7)** What is Tokenizer **.**
**8) Explain window size parameter in detail for CBOW model.**
**9) Explain Embedding and Lmbda layer from keras**
**10) What is yield()**

**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(positive=['Your word'])

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

**Conclusion: Explain how Neural network is useful for CBOW text analysis.**

# 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

**Theory : 1)**What is **Transfer learning** ?
2) What are pretrained Neural Network models ?
3) Explain Pytorch library in short.
4) What are advantages of Transfer learning.
5) What are applications of Transfer learning.
**6)** Explain Caltech 101 images dataset.
7)Explain Imagenet dataset **.**
8)list down basic steps for transfer learning.
9) What is Data augmentation?
10) How and why Data augmentation is done related to transfer learning?
11) Why preprocessing is needed on inputdata in Transfer learning.
12) What is PyTorch Transforms module.Explain following commands w.r.t it :

```
Compose([
        RandomResizedCrop(size=256, scale=(0.8, 1.0)),
        RandomRotation(degrees=15),
        ColorJitter(),
        RandomHorizontalFlip(),
        CenterCrop(size=224),  # Image net standards
   .ToTensor(),
        Normalize
```

13) Explain the Validation Transforms steps with Pytorch Transforms .
14) Explain VGG-16 model from Pytorch

**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: Explain how Transfer training increases the accuracy of Object detection**

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