INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY

INNOVATION & LEADERSHIP

www.isquareit.edu.in

DEPARTMENT OF INFORMATION TECHNOLOGY

LAB MANUAL

COURSE: LAB PRACTICE IV

COURSE CODE:414447

CLASS: BEIT (2019 PATTERN)

SEMESTER-I

COURSE CORDINATOR: Dr. Jyoti Surve

Vision of the Institute:

To be a premier academic institution that fosters diversity, value-added education and research, leading to sustainable innovations and transforming learners into leaders.

Mission of the Institute:

· To strive for academic excellence, knowledge enhancement, and critical thinking capabilities by adopting innovative and dynamic teaching-learning pedagogies

· To enrich and leverage interactions and associations through Industry-Academia partnerships

· To groom students so as to make them lifelong learners by helping them imbibe professional, entrepreneurial and leadership qualities

· To embrace an environment that allows all stakeholders to benefit from the technology-enabled processes and systems

Vision of the Department:

To be recognized as an ingenious department that provides value added education with technology excellence to transform students into lifelong learners and ethical professionals.

Mission of the Department:

- To provide excellent academics with positive atmosphere in the domain of Information Technology.
- To impart new technologies with innovative and interactive teaching learning.
- To encourage students for higher studies, skilled professionals and future entrepreneur.
- To groom the student with ethical and social values.

Program Educational Objectives (PEOs):

- PEO1: Possess strong fundamental concepts in mathematics, science, engineering and technology to be able to adapt to dynamic technological challenges and become effective professionals.

- PEO2: Acquire technical and analytical skills in the field of information technology so as to be able to analyse, design and implement efficient solutions to complex engineering problems with innovative approaches.

- PEO3: Develop proficiency in communication skills combined with domain specific competencies leading to success in higher education and as technologist and/or entrepreneur.

- PEO4: Be committed to abiding by ethical practices and become socially conscientious individuals

Program Outcomes (POs)

PO1: An ability to apply knowledge of mathematics, computing, science, engineering and technology.

PO2: An ability to define a problem and provide a systematic solution with the help of conducting experiments, analyzing the problem and interpreting the data.

PO3: An ability to design, implement, and evaluate a software or a software/hardware system, component, or process to meet desired needs within realistic constraints.

PO4: An ability to identify, formulates, and provides systematic solutions to complex engineering/Technology problems.

PO5: An ability to use the techniques, skills, and modern engineering technology tools, standard processes necessary for practice as a IT professional.

PO6: 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.

PO7: An ability to analyze and provide solution for the local and global impact of information technology on individuals, organizations and society.

PO8: An ability to understand professional, ethical, legal, security and social issues and responsibilities.

PO9: An ability to function effectively as an individual or as a team member to accomplish a desired goal(s).

PO10: An ability to communicate effectively in engineering community at large by means of effective presentations, report writing, paper publications, demonstrations.

PO11: An ability to understand engineering, management, financial aspects, performance, optimizations and time complexity necessary for professional practice.

PO12: 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 extracurricular activities.

Program Specific Outcomes (PSO)

PSO1: An ability to apply the theoretical concepts and practical knowledge of Information Technology in analysis, design, development and management of information processing systems and applications in the interdisciplinary domain.

PSO2: An ability to analyse a problem and identify and define the computing infrastructure and operations requirements appropriate to its solution. IT graduates should be able to work on large-scale computing systems.

PSO3: An understanding of professional, business and business processes, ethical, legal, security and social issues and responsibilities.

PSO4: Practice communication and decision-making skills through the use of appropriate technology and be ready for professional responsibilities

414447: Lab Practice IV

Teaching Scheme:	**Credit Scheme:**	**Examination Scheme:**
Practical (PR):02 hrs/week	01 credits	PR: 25 Marks TW: 25 Marks

COURSE EDUCATIONAL OBJECTIVES(CEO)

Course Objective	**Description**
CEO407.1	To be able to formulate deep learning problems corresponding to different applications.
CEO407.2	To be able to apply deep learning algorithms to solve problems of moderate complexity.
CEO407.3	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 OUTCOME (CO)

Course Outcome	**Description**
CO407.1	Learn and Use various Deep Learning tools and packages.
CO407.2	Build and train a deep Neural Network models for use in various applications
CO407.3	Apply Deep Learning techniques like CNN, RNN Auto encoders to solve real word Problems
CO407.4	Evaluate the performance of the model build using Deep Learning

List of Practical Assignments

Sr. No.	**Title of the Experiment/Assignment**	**CEO**	**CO**	**BTL**
1	Study of Deep learning Packages: Tensorflow, Keras, Theano and PyTorch	CEO407.1,CEO407.2	CO407.1	BTL 2
2	Implementing Feedforward neural networks with Keras and TensorFlow	CEO407.2,CEO407.3	CO407.2, CO407.4	BTL 4
3	Build the Image classification model	CEO407.2,CEO407.3	CO407.3, CO407.4	BTL 4
4	Use Autoencoder to implement anomaly detection	CEO407.2,CEO407.3	CO407.2	BTL 4
5	Implement the Continuous Bag of Words (CBOW) Model	CEO407.2,CEO407.3	CO407.3	BTL 4
6	Object detection using Transfer Learning of CNN architectures	CEO407.2,CEO407.3	CO407.4	BTL 4

INDEX

Sr.No.	Title of the Assignment	Page No.
1	Study of Deep learning Packages: Tensorflow, Keras, Theano and PyTorch	
2	Implementing Feedforward neural networks with Keras and TensorFlow	
3	Build the Image classification model	
4	Use Autoencoder to implement anomaly detection	
5	Implement the Continuous Bag of Words (CBOW) Model	
6	Object detection using Transfer Learning of CNN architectures	

Assignment No.1

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

Objectives:

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

Theory:

1. Introduction to TENSORFLOW

TensorFlow is an end-to-end open-source platform for machine learning. TensorFlow is a rich system for managing all aspects of a machine learning system; however, this class focuses on using a particular TensorFlow API to develop and train machine learning models

TensorFlow APIs are arranged hierarchically, with the high-level APIs built on the low-level APIs. Machine learning researchers use the low-level APIs to create and explore new machine learning algorithms. In this class, you will use a high-level API named tf.keras to define and train machine learning models and to make predictions. tf.keras is the TensorFlow variant of the open-source Keras API.
The following figure shows the hierarchy of TensorFlow toolkits:

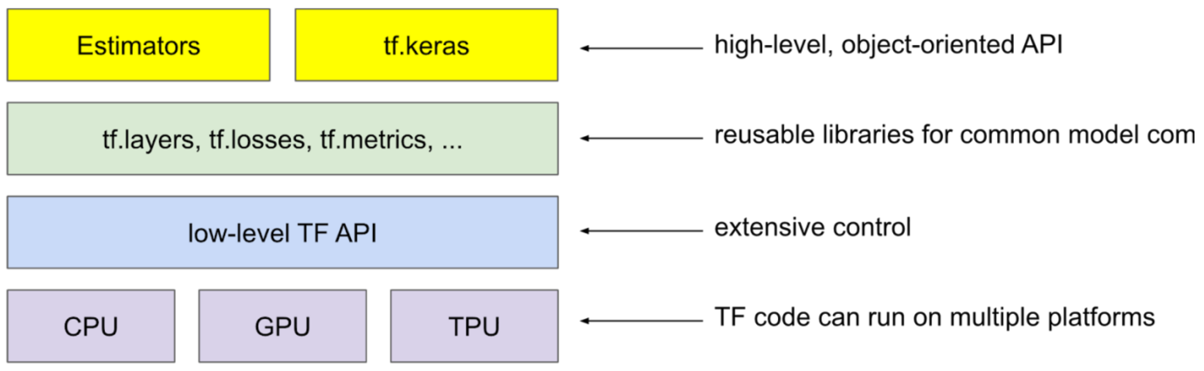
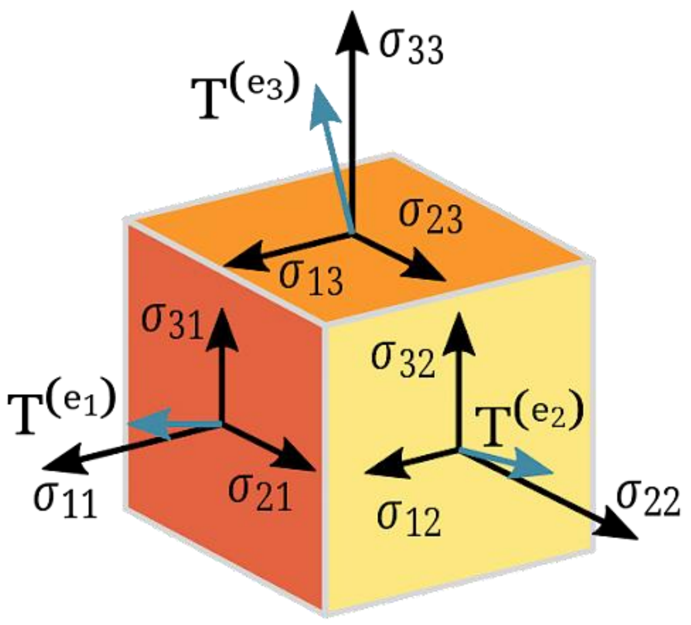

Estimators
tf.keras
high-level, object-oriented API
tf.layers, tf.losses, tf.metrics, ...
reusable libraries for common model com
low-level TF API
extensive control
CPU
GPU
TPU
TF code can run on multiple platforms

Figure 1. TensorFlow toolkit hierarchy.

Why is TensorFlow used in deep learning?
As the training of the models in deep learning takes extremely long because of the large amount of data, using TensorFlow **makes it much easier to write the code for GPUs or CPUs and then execute it in a distributed manner**
TensorFlow accepts data in the form of multi-dimensional arrays of higher dimensions called tensors. Multi-dimensional arrays are very handy in handling large amounts of data.
TensorFlow works on the basis of data flow graphs that have nodes and edges. As the execution mechanism is in the form of graphs, it is much easier to execute TensorFlow code in a distributed manner across a cluster of computers while using GPUs.
Tensor
Tensor forms the core framework of TensorFlow. All the computations in TensorFlow involve tensors. It is a matrix of n-dimensions that represents multiple types of data. A tensor can be the result of a computation or it can originate from the input data.



Graphs
Graphs describe all the operations that take place during the training. Each operation is called an op node and is connected to the other. The graph shows the op nodes and the connections between the nodes, but it does not display values.

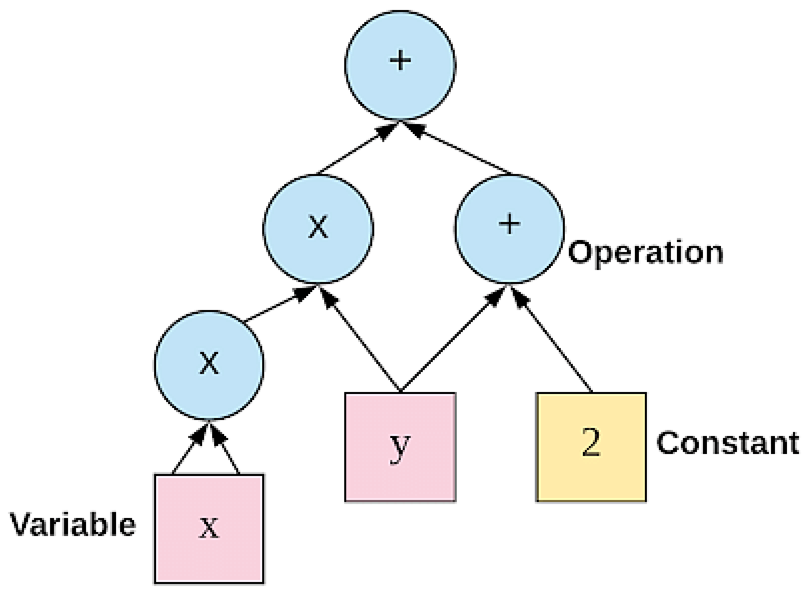
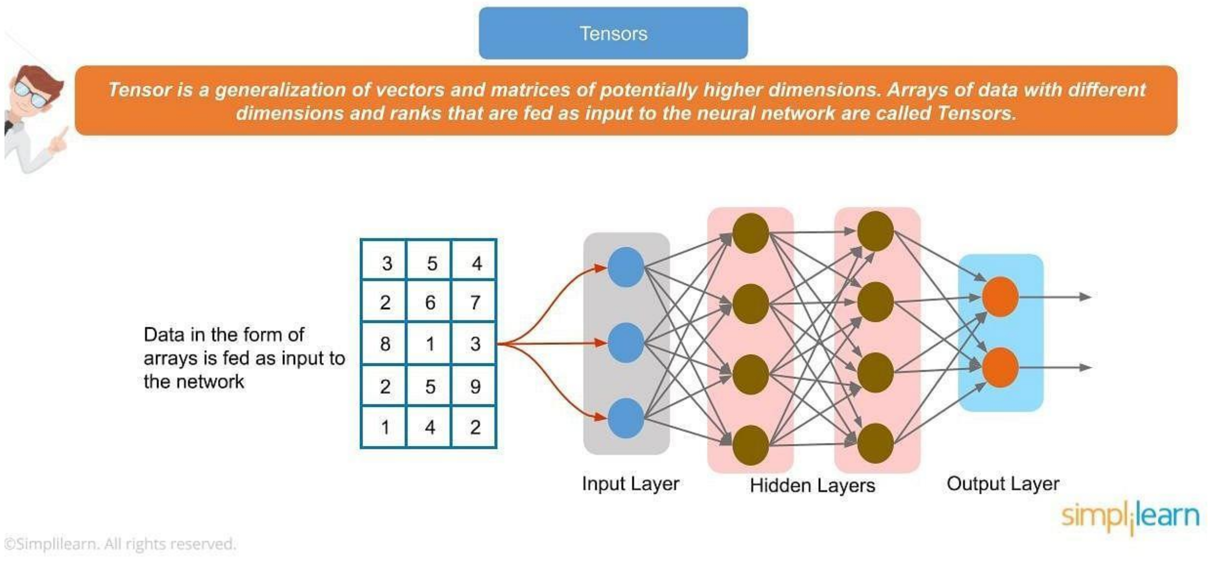


Fig: Graphs (source)

What are Tensors?
Tensor is a generalization of vectors and matrices of potentially higher dimensions. Arrays of data with varying dimensions and ranks that are fed as input to the neural network are called tensors.
For deep learning, especially in the training process, you will have large amounts of data that exist in a very complicated format. It helps when you are able to put, use, or store it in a compact way, which tensors provide, even if they appear in multi-dimensional arrays. When the data is stored in tensors and fed into the neural network, the output we get is as shown below:

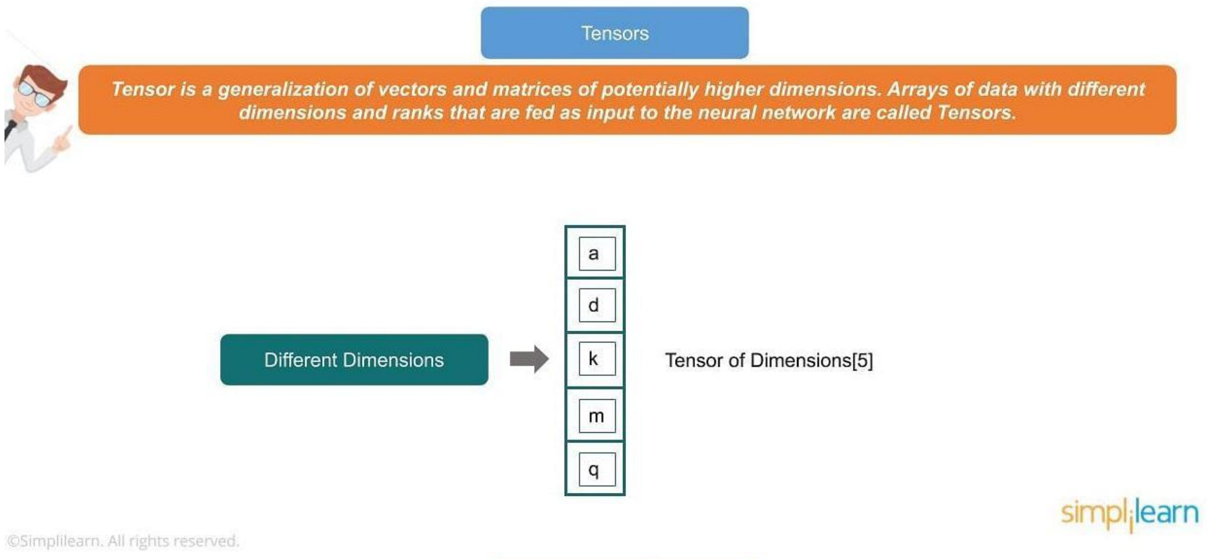

Tensors
Data in the form of arrays is fed as input to the network
3 5 4
2 6 7
8 1 3
2 5 9
1 4 2
Input Layer
Hidden Layers
Output Layer
simplilearn
©Simplilearn. All rights reserved.

Tensor is a generalization of vectors and matrices of potentially higher dimensions. Arrays of data with different dimensions and ranks that are fed as input to the neural network are called Tensors.

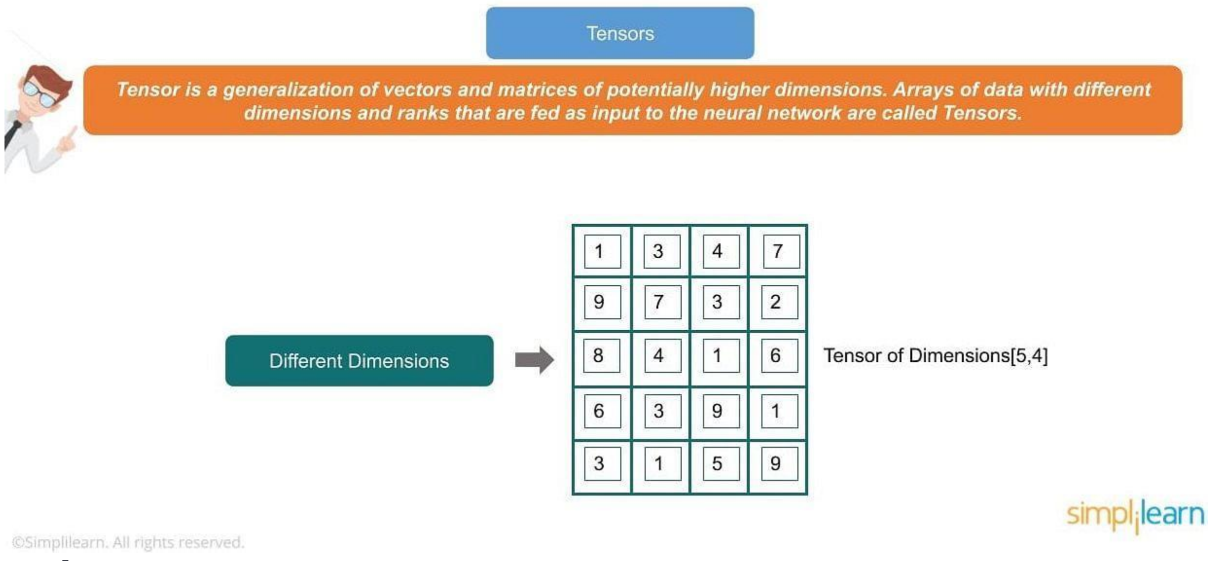
There are some terms associated with tensors that we need to familiarize ourselves with:

Dimension

Dimension is the size of the array elements. Below you can take a look at various types of dimensions:



Tensor is a generalization of vectors and matrices of potentially higher dimensions. Arrays of data with different dimensions and ranks that are fed as input to the neural network are called Tensors.



Tensor is a generalization of vectors and matrices of potentially higher dimensions. Arrays of data with different dimensions and ranks that are fed as input to the neural network are called Tensors.

Ranks

Tensor ranks are the number of dimensions used to represent the data. For example:

Rank 0 - When there is only one element. We also call this as a scalar.

Example: s = [2000]

Rank 1 - This basically refers to a one-dimensional array called a vector.
Example: v = [10, 11, 12]

2 Introduction to Keras

Keras is a high-level, deep learning API developed by Google for implementing neural networks. It is written in Python and is used to make the implementation of neural networks easy. It also supports multiple backend neural network computation.

Keras is relatively easy to learn and work with because it provides a python frontend with a high level of abstraction while having the option of multiple back-ends for computation purposes. Keras allows you to switch between different back ends. The frameworks supported by Keras are:

- Tensorflow
- Theano
- PlaidML
- MXNet
- CNTK (Microsoft Cognitive Toolkit)

Out of these five frameworks, TensorFlow has adopted Keras as its official high-level API. Keras is embedded in TensorFlow and can be used to perform deep learning fast as it provides inbuilt modules for all neural network computations.

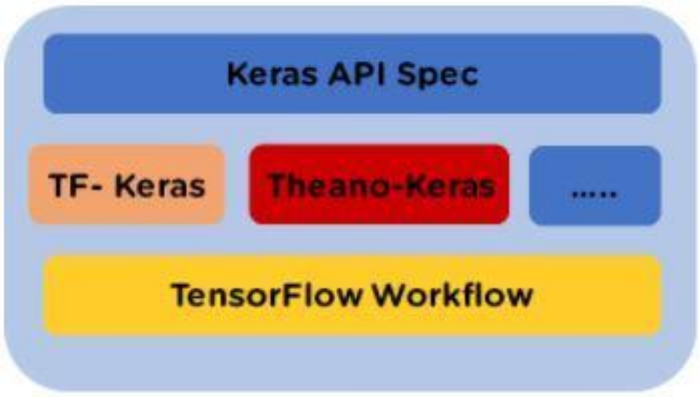


Figure 2: Keras Backend

Why Do We Need Keras?

- Keras is an API that was made to be easy to learn for people. Keras was made to be simple. It offers consistent & simple APIs, reduces the actions required to implement common code, and explains user error clearly.
- Prototyping time in Keras is less. This means that your ideas can be implemented and deployed in a shorter time. Keras also provides a variety of deployment options depending on user needs.

How to Build a Model in Keras?

The below diagram shows the basic steps involved in building a model in Keras:

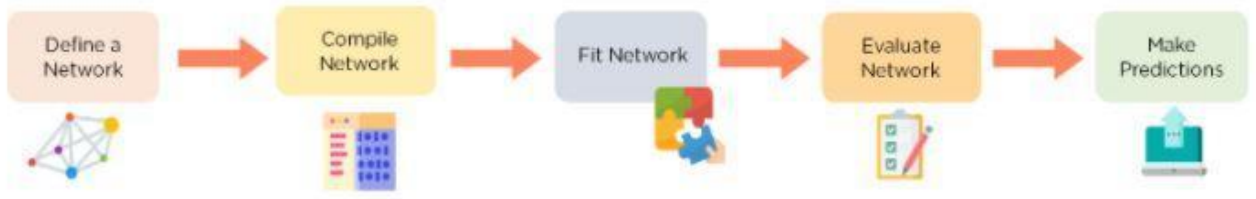


Figure 3: Building a model

1. Define a network: In this step, you define the different layers in our model and the connections between them. Keras has two main types of models: Sequential and Functional models. You choose which type of model you want and then define the dataflow between them.
2. Compile a network: To compile code means to convert it in a form suitable for the machine to understand. In Keras, the model.compile() method performs this function. To compile the model, we define the loss function which calculates the losses in our model, the optimizer which reduces the loss, and the metrics which is used to find the accuracy of our model.
3. Fit the network: Using this, we fit our model to our data after compiling. This is used to train the model on our data.
4. Evaluate the network: After fitting our model, we need to evaluate the error in our model.

5. Make Predictions: We use model.predict() to make predictions using our model on new data.

- **Applications of Keras**

- Keras is used for creating deep models which can be productized on smartphones.
- Keras is also used for distributed training of deep learning models.
- Keras is used by companies such as Netflix, Yelp, Uber, etc.
- Keras is also extensively used in deep learning competitions to create and deploy working models, which are fast in a short amount of time.

3. Introduction to Theano

Theano is a Python library that lets you define mathematical expressions used in Machine Learning, optimize these expressions and evaluate those very efficiently by decisively using GPUs in critical areas. It can rival typical full C-implementations in most of the cases.

Theano was written at the LISA lab with the intention of providing rapid development of efficient machine learning algorithms. It is released under a BSD license.

Theano can be installed on Windows, MacOS, and Linux. The installation in all the cases is trivial. Before you install Theano, you must install its dependencies. The following is the list of dependencies −

- Python
- NumPy − Required
- SciPy − Required only for Sparse Matrix and special functions
- BLAS − Provides standard building blocks for performing basic vector and matrix operations

The optional packages that you may choose to install depending on your needs are −

- nose: To run Theano’s test-suite
- Sphinx − For building documentation
- Graphiz and pydot − To handle graphics and images
- NVIDIA CUDA drivers − Required for GPU code generation/execution

Theano facilitates defining mathematical expressions used in ML development. Such expressions generally involve Matrix Arithmetic, Differentiation, Gradient Computation, and so on.

Theano first builds the entire Computational Graph for your model. It then compiles it into highly efficient code by applying several optimization techniques on the graph. The compiled code is injected into Theano runtime by a special operation called **function** available in Theano.

4.Introduction to Pytorch

PyTorch is defined as an open source machine learning library for Python. It is used for applications such as natural language processing. It is initially developed by Facebook artificial-intelligence research group, and Uber’s Pyro software for probabilistic programming which is built on it.

Features

The major features of PyTorch are mentioned below −

Easy Interface − PyTorch offers easy to use API; hence it is considered to be very simple to operate and runs on Python. The code execution in this framework is quite easy.

Python usage − This library is considered to be Pythonic which smoothly integrates with the Python data science stack. Thus, it can leverage all the services and functionalities offered by the Python environment.

Computational graphs − PyTorch provides an excellent platform which offers dynamic computational graphs. Thus a user can change them during runtime. This is highly useful when a developer has no idea of how much memory is required for creating a neural network model.

PyTorch is known for having three levels of abstraction as given below −

- Tensor − Imperative n-dimensional array which runs on GPU.
- Variable − Node in computational graph. This stores data and gradient.
- Module − Neural network layer which will store state or learnable weights.

TensorFlow vs. PyTorch

We shall look into the major differences between TensorFlow and PyTorch below −

PyTorch	TensorFlow
PyTorch is closely related to the lua-based Torch framework which is actively used in Facebook.	TensorFlow is developed by Google Brain and actively used at Google.

PyTorch is relatively new compared to other competitive technologies.	TensorFlow is not new and is considered as a to-go tool by many researchers and industry professionals.
PyTorch includes everything in an imperative and dynamic manner.	TensorFlow includes static and dynamic graphs as a combination.
Computation graph in PyTorch is defined during runtime.	TensorFlow does not include any run time option.
PyTorch includes deployment features for mobile and embedded frameworks.	TensorFlow works better for embedded frameworks.

Algorithms/Pseudo code:

Sample Programs:

Kindly run it on Google colab

/*** **SAMPLE PROGRAM USING TENSORFLOW**/**

```
!pip install tensorflow
# import
import tensorflow as tf
import numpy as np
a = tf.constant(15)
b = tf.constant(20)
print(a+b)
# input
x = np.random.rand(100).astype(np.float32)
print(x)
# output - observed
y = x * 0.2 + 0.2
# Weight
W = tf.Variable(tf.random.normal([1]))
# bias
b = tf.Variable(tf.zeros([1]))
# Create a function for MSE - mean squared error
def mse_loss():
  ypred = W * x + b
  loss = tf.reduce_mean(tf.square(ypred-y))
  return loss
# Optimizer
optimizer = tf.keras.optimizers.Adam()
# Iterations
for step in range(5000):
  optimizer.minimize(mse_loss,var_list=[W,b])
  if step % 500 == 0:
    print(step, W.numpy(), b.numpy())
```

/*** **SAMPLE PROGRAM USING KERAS**/**

```
# !pip install tensorflow
# Step 1 - Load the dataset
from numpy import loadtxt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
## INPUT Variables ##
# x1 - Number of times pregnant
# x2 - plasma glucose
# x3 - diastolic blood pressure
# x4 - Triceps skin fold thickness
# x5 - 2-hour serum insulin
# x6 - bmi
# x7 - diabetes pedigree function
# x8 - age (yrs)
## Output Variable ##
# Class Variable - 0 or 1

dataset = loadtxt('/content/pima-indians-diabetes.csv',delimiter=',')
dataset
# [:,:] - first : is range of rows and second : is columns
# [start:end] - begins at start, ends at end-1
x = dataset[:,0:8]
print(type(x))
print(x.shape)
y = dataset[:,8]
print(y)
# Step 2 - Creating or define the Keras Model
# Sequential Model
# Layer1 -> Layer2 -> Layer3

model = Sequential()
# The model expects row of data with 8 variables
# 12 = nodes
model.add(Dense(12, input_shape=(8,), activation='relu'))
# Hidden Layer
# 8 = nodes
model.add(Dense(8, activation='relu'))
# Output layer
model.add(Dense(1,activation='sigmoid'))
# Step 3 - Compile the Keras model
# loss (error)
# optimizer (adam)
# metrics = accuracy
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
 Step 4 - Fit / Train the model
#1 = Epochs - number of iterations / passes
#2 - Batch - sample data
model.fit(x,y, epochs=150, batch_size=10)
# Step 5 - evaluate the model
model.evaluate(x,y)
```

/*** **SAMPLE PROGRAM USING THEANO**/**

```
# Theano
# Mathematical Operations
# Arrays - Multi-dimensional
# Installation
!pip install theano
import theano
from theano import *
import theano.tensor as T
import numpy as np
import pandas as pd
from theano import function

# scalar variables
v1 = T.dscalar()
v2 = T.scalar()
# subtraction
sres = v1-v2

#add
ares = v1+v2
 convert the results into functions
calcsres = theano.function([v1,v2],sres)
calcares = theano.function([v1,v2],ares)
calcares(12,23)
calcsres(13,12)
x = T.dmatrix('x')
y = T.dmatrix('y')

# addition
z = x+y
func = function([x,y],z)
m1 = [
    [1,2],
    [3,4]
]
m2 = [
    [4,5],
    [6,7]
]
func(m1,m2)
# element wise sum
# 00 -> [1+4] -> 5
```

/*** SAMPLE PROGRAM USING PYTORCH**/

```
# Two major objectives of PyTorch
#1. Replacement of Numpy to use the power of GPUs and other accelerators
#2. Automatic Differentiation Library helps in implementation of Neural Networks


# Feed Forward --> first stage
# Back Propagation --> second stage (differentiation)
# Tensors - Special Data Structures
# similar to an array, matrices
# similar to Numpy ndarrays
import torch
import numpy as np
# Tensor Initialization
### multiple ways
### 1 - using data
data = [
   [1,2],
   [3,4]
]
x_data = torch.tensor(data)

print(type(x_data))
### 2 - using numpy array
np_array = np.array(data)
x_np = torch.from_numpy(np_array)
print(x_np)
print(type(x_np))
### 3 - using another tensor
x_ones = torch.ones_like(x_data)
print("One Tensor: \n",x_ones)
x_rand = torch.rand_like(x_data,dtype=torch.float)
print(x_rand)
#### more ways to create tensors
shape = (2,3)
random_tensor = torch.rand(shape)
print(random_tensor)
print(type(random_tensor))
ones_tensor = torch.ones(shape)
print(ones_tensor)
print(type(ones_tensor))
zeros_tensor = torch.zeros(shape)
print(zeros_tensor)
print(type(zeros_tensor))
tensor = torch.rand(3,4)
print(tensor)
tensor.shape
tensor.dtype
tensor.device
# Tensor Operations
if torch.cuda.is_available():
  tensor = tensor.to('cuda')
  print("Device tensor is stored in ", tensor.device)
# Indexing, Slicing
tensor = torch.ones(4,4)
```

```
print(tensor)
print(tensor)
tensor1 = torch.zeros(4,4)
print(tensor1)
tensor2 = torch.cat([tensor,tensor1])
print(tensor2)
# Multiply Operation
tensor.mul(tensor1)
tensor * tensor1
tensor.T
# inplace - change the original tensor
tensor.add_(3)
print(tensor)
# from tensor to numpy
t = torch.ones(5)
print(t)
n = t.numpy()
print(n)
print(type(n))
```

Conclusion:

Thus we have studied different packages like tensor flow keras, pytorch, theano and their associated libraries

FAQ:

1. What is tensor?

2. What is dimension and Ranks in tensor flow?

3. How to build model in keras?

4. What is function operation in Theano?

5.How Pytorch is different from Tensor flow?

Output:

Assignment No.2

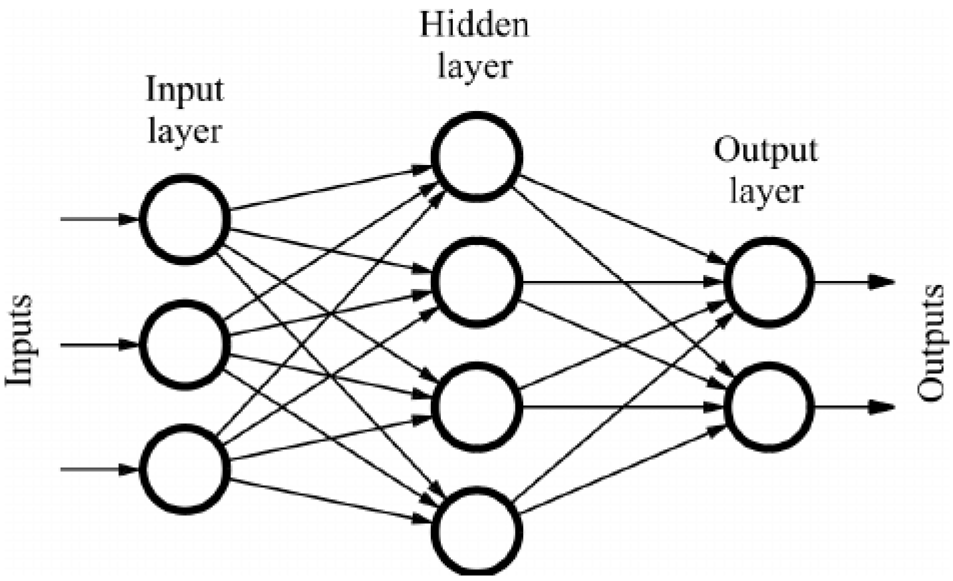
Aim: Implement Feed forward neural networks with Keras and TensorFlow

Objectives:

1. Study of deep learning Packages: Tensorflow, Keras.
2. Perform following operations on given dataset
   - **a.** Import the necessary packages
   - **b.** Load the training and testing data (MNIST/CIFAR10)
   - **c.** Define the network architecture using Keras
   - **d.** Train the model using SGD
   - **e.** Evaluate the network
   - **f.** Plot the training loss and accuracy

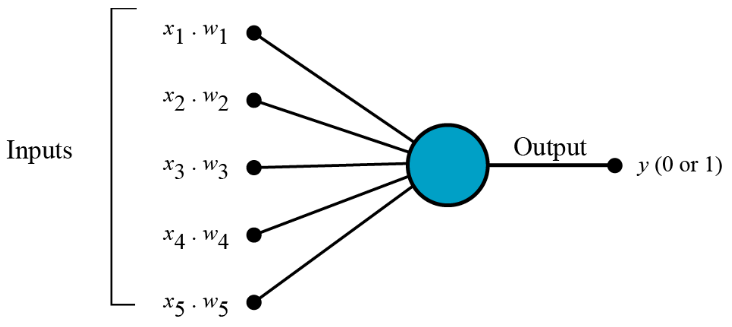
Theory:

A Feed Forward Neural Network is an artificial neural network in which the connections between nodes does not form a cycle. The opposite of a feed forward neural network is a recurrent neural network, in which certain pathways are cycled. The feed forward model is the simplest form of neural network as information is only processed in one direction. While the data may pass through multiple hidden nodes, it always moves in one direction and never backwards.



How does a Feed Forward Neural Network work?

A Feed Forward Neural Network is commonly seen in its simplest form as a single layer perceptron. In this model, a series of inputs enter the layer and are multiplied by the weights. Each value is then added together to get a sum of the weighted input values. If the sum of the values is above a specific threshold, usually set at zero, the value produced is often 1, whereas if the sum falls below the threshold, the output value is -1. The single layer perceptron is an important model of feed forward neural networks and is often used in classification tasks. Furthermore, single layer perceptrons can incorporate aspects of machine learning. Using a property known as the delta rule, the neural network can compare the outputs of its nodes with the intended values, thus allowing the network to adjust its weights through training in order to produce more accurate output values. This process of training and learning produces a form of a gradient descent. In multi-layered perceptrons, the process of updating weights is nearly analogous, however the process is defined more specifically as back-propagation. In such cases, each hidden layer within the network is adjusted according to the output values produced by the final layer.



Load the training and testing data (MNIST/CIFAR10)

1. MNIST

Use the *full* MNIST dataset, consisting of 70,000 data points (7,000 examples per digit). Each data point is represented by a 784-d vector, corresponding to the (flattened) *28×28* images in the MNIST dataset. Our goal is to train a neural network (using Keras) to obtain > *90%* accuracy on this dataset.

MNIST (Modified National Institute of Standards and Technology) is a well-known dataset used in Computer Vision that was built by Yann Le Cun et. al. It is composed of images that are **handwritten digits (0-9).**

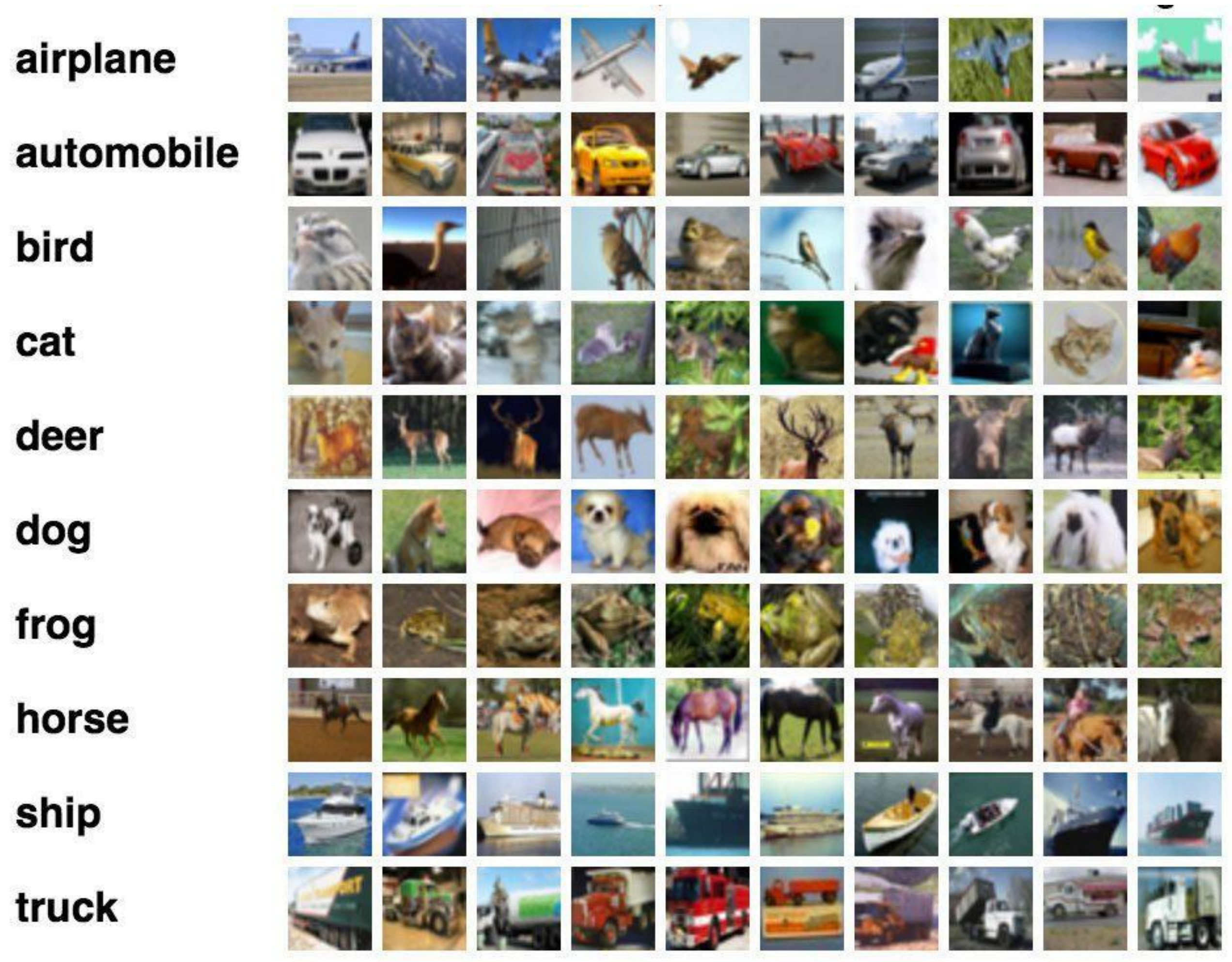
you can download it from below URL

http://yann.lecun.com/exdb/mnist/

2.CIFAR-10

When it comes to computer vision and machine learning, the MNIST dataset is the classic definition of a “benchmark” dataset, one that is too easy to obtain high accuracy results on, and not representative of the images we’ll see in the real world.

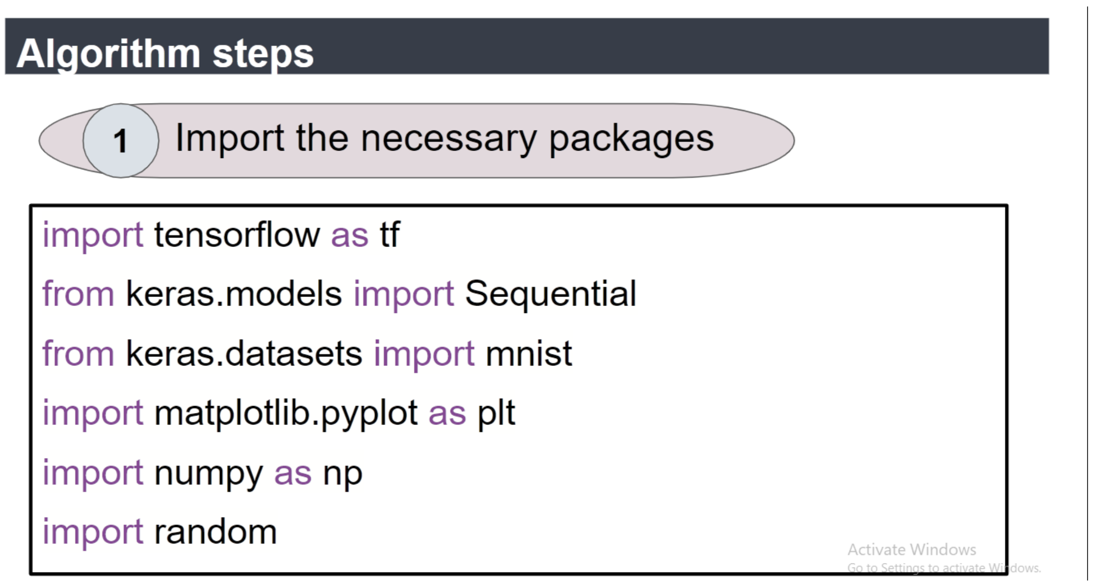
For a more challenging benchmark dataset, we commonly use CIFAR-10, a collection of 60,000, 32×32 RGB images, thus implying that each image in the dataset is represented by 32×32×3 = 3,072 integers. As the name suggests, CIFAR-10 consists of 10 classes, including airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. A sample of the CIFAR-10 dataset for each class can be seen in **Figure 2**.



Each class is evenly represented with 6,000 images per class. When training and evaluating a machine learning model on CIFAR-10, it’s typical to use the predefined data splits by the authors and use 50,000 images for training and 10,000 for testing

As we’ll find out, using Keras to build our network architecture is *substantially easier* than our pure Python version. In fact, the actual network architecture will only occupy *four lines of code* — the rest of the code in this example simply involves loading the data from disk, transforming the class labels, and then displaying the results.

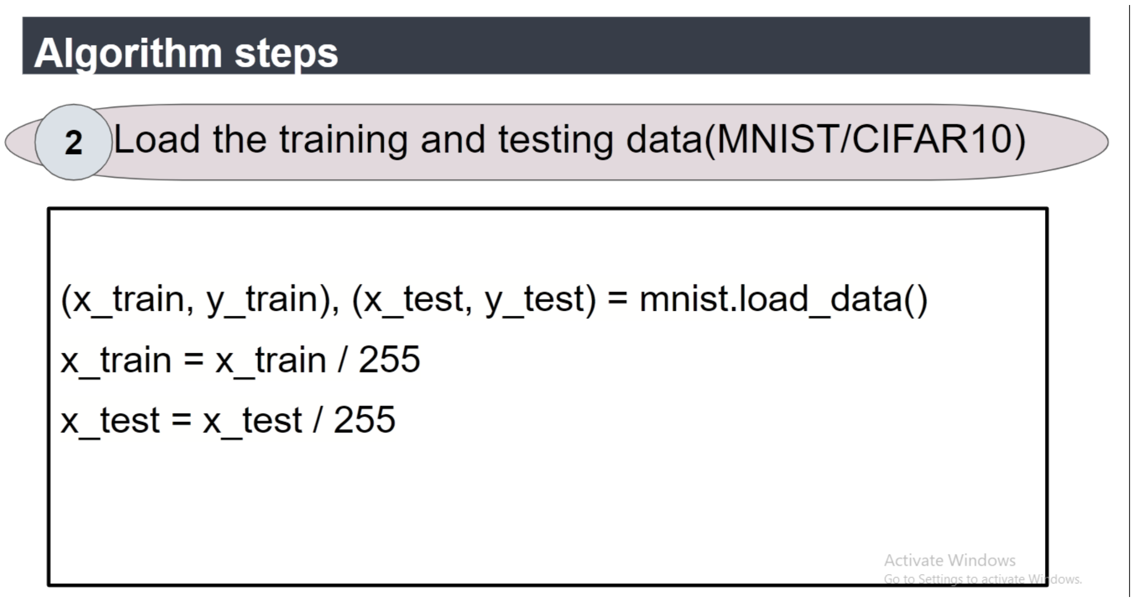
Algorithm/Pseudo code:



Algorithm steps

1 Import the necessary packages

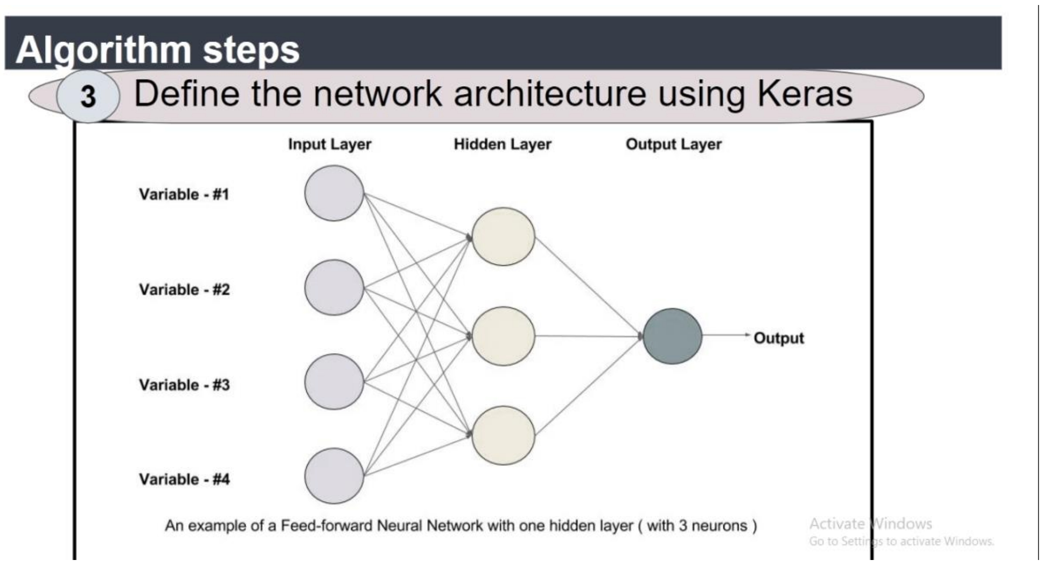
```
import tensorflow as tf
from keras.models import Sequential
from keras.datasets import mnist
import matplotlib.pyplot as plt
import numpy as np
import random
```



Algorithm steps

2 Load the training and testing data(MNIST/CIFAR10)

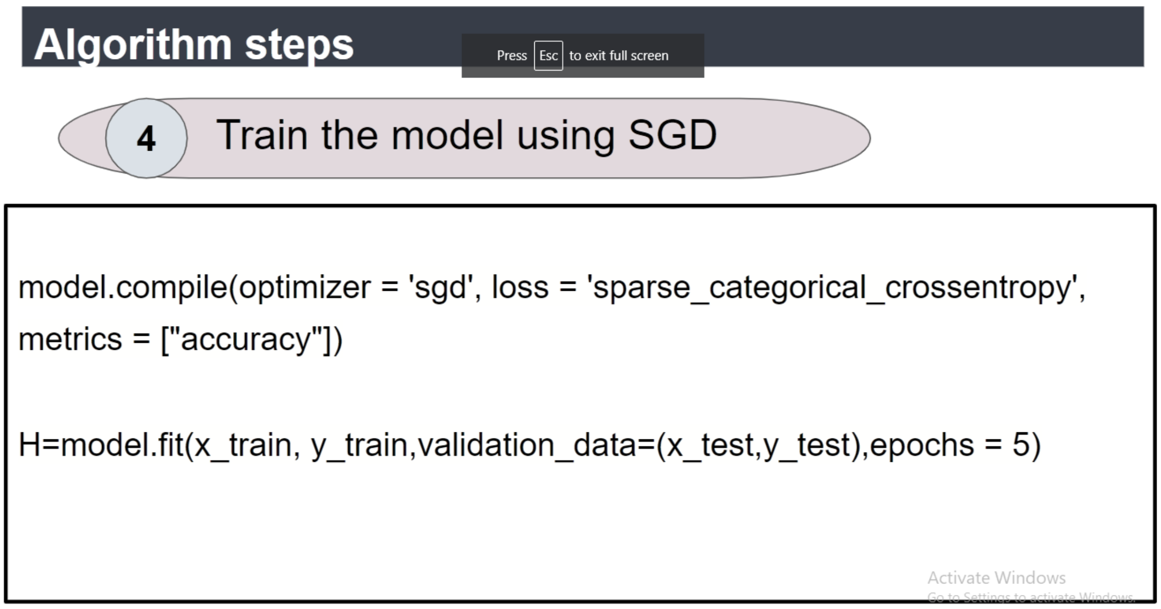
```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train / 255
x_test = x_test / 255
```



Algorithm steps

3 Define the network architecture using Keras

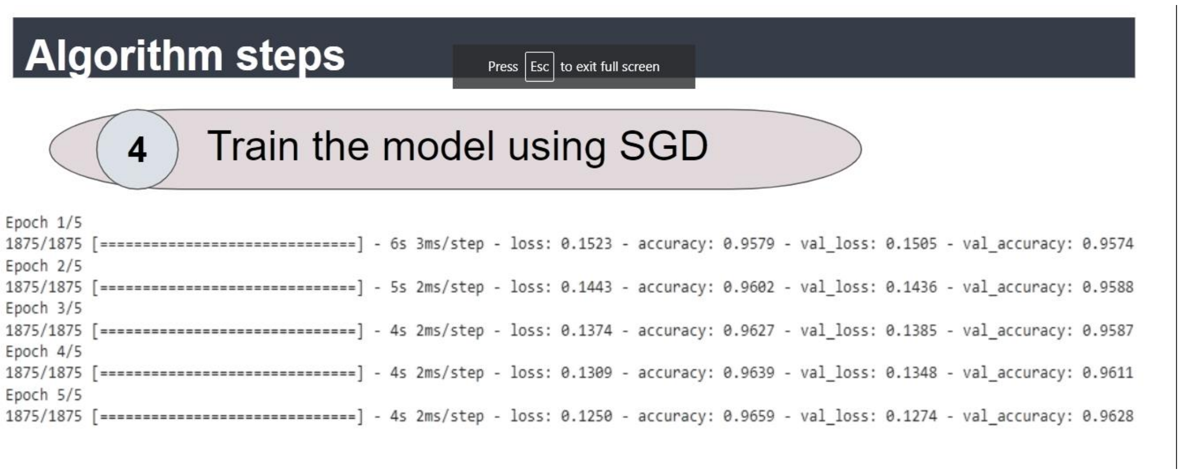
An example of a Feed-forward Neural Network with one hidden layer (with 3 neurons)


Press Esc to exit full screen
Activate Windows
Go to Settings to activate Windows.

Algorithm steps

4 Train the model using SGD

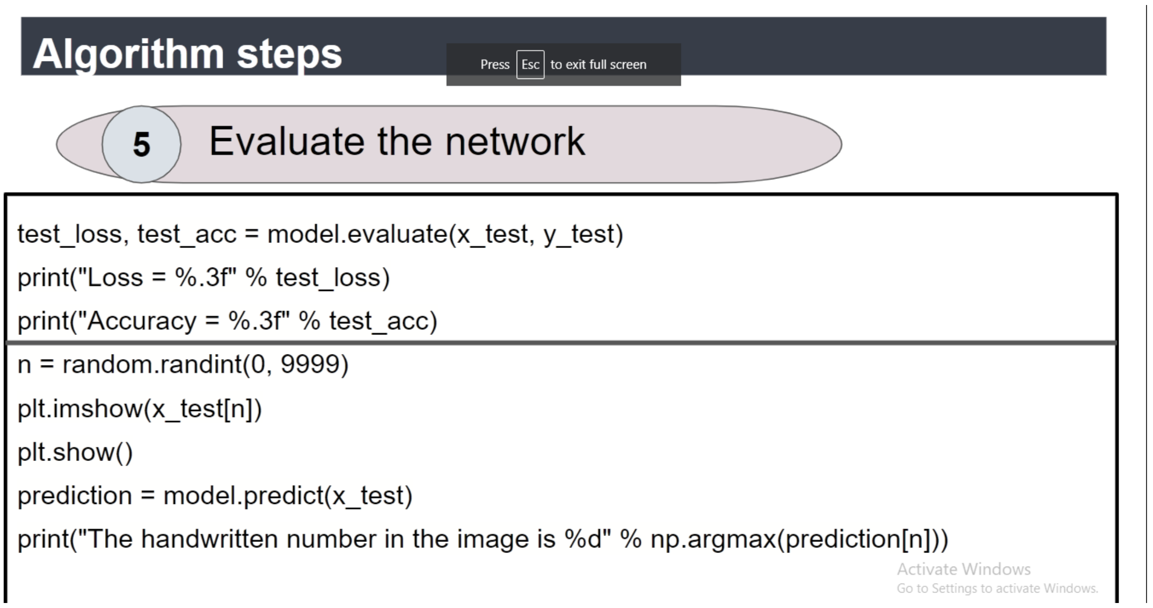
```
model.compile(optimizer = 'sgd', loss = 'sparse_categorical_crossentropy',
metrics = ["accuracy"])

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



Algorithm steps

4 Train the model using SGD

```
Epoch 1/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1523 - accuracy: 0.9579 - val_loss: 0.1505 - val_accuracy: 0.9574
Epoch 2/5
1875/1875 [==============================] - 5s 2ms/step - loss: 0.1443 - accuracy: 0.9602 - val_loss: 0.1436 - val_accuracy: 0.9588
Epoch 3/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1374 - accuracy: 0.9627 - val_loss: 0.1385 - val_accuracy: 0.9587
Epoch 4/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1309 - accuracy: 0.9639 - val_loss: 0.1348 - val_accuracy: 0.9611
Epoch 5/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1250 - accuracy: 0.9659 - val_loss: 0.1274 - val_accuracy: 0.9628
```

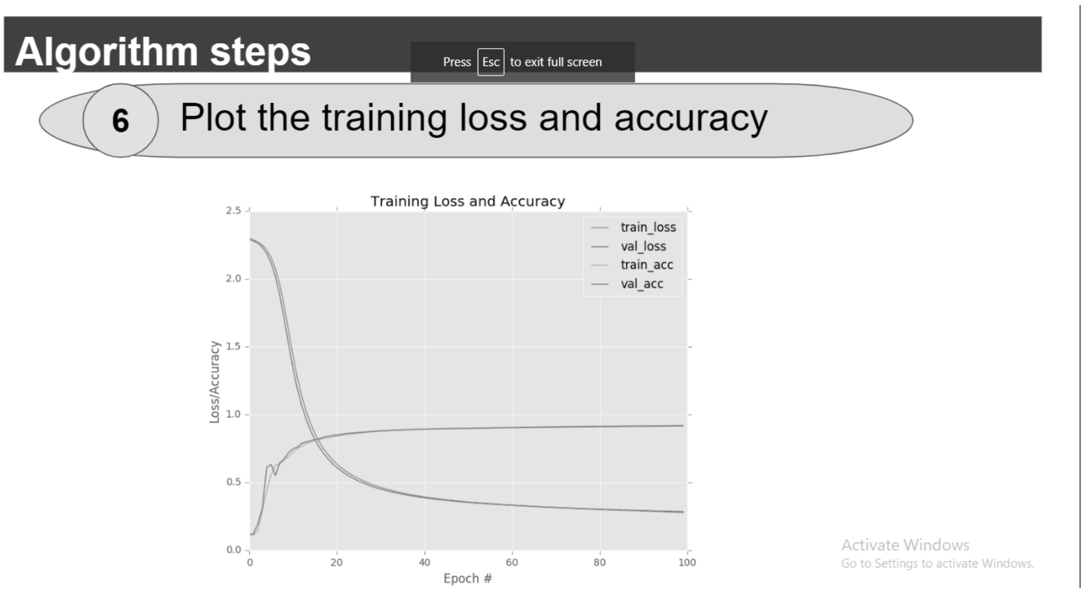

Press Esc to exit full screen
Activate Windows
Go to Settings to activate Windows.

Algorithm steps

5 Evaluate the network

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

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


Algorithm steps

6 Plot the training loss and accuracy

/******* SAMPLE PROGRAM ********/

```
# import the necessary packages
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import classification_report
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.datasets import mnist
from tensorflow.keras import backend as K
import matplotlib.pyplot as plt
import numpy as np
import argparse


# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-o", "--output", required=True,
help="path to the output loss/accuracy plot")
args = vars(ap.parse_args())
# grab the MNIST dataset (if this is your first time using this
# dataset then the 11MB download may take a minute)
print("[INFO] accessing MNIST...")
((trainX, trainY), (testX, testY)) = mnist.load_data()
# each image in the MNIST dataset is represented as a 28x28x1
# image, but in order to apply a standard neural network we must
# first "flatten" the image to be simple list of 28x28=784 pixels
trainX = trainX.reshape((trainX.shape[0], 28 * 28 * 1))
testX = testX.reshape((testX.shape[0], 28 * 28 * 1))
# scale data to the range of [0, 1]
trainX = trainX.astype("float32") / 255.0
testX = testX.astype("float32") / 255.0
```

```
# convert the labels from integers to vectors
lb = LabelBinarizer()
trainY = lb.fit_transform(trainY)
testY = lb.transform(testY)
```

consider the label 3 and we wish to binarize/one-hot encode it — the label 3 now becomes:
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0]

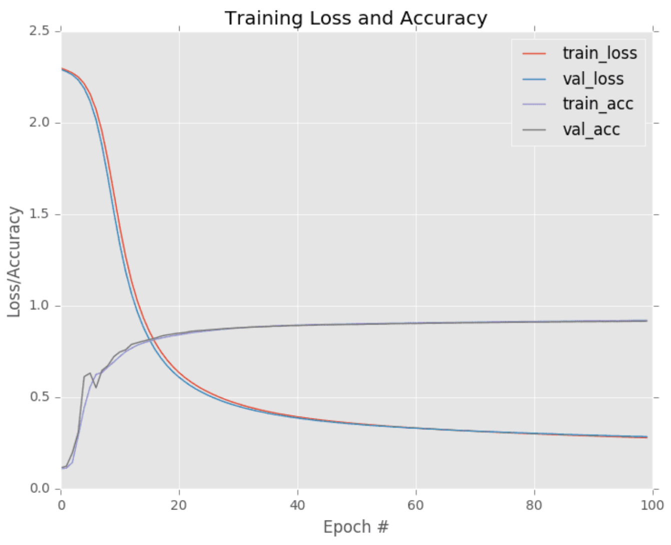
Here is a second example, this time with the label 1 binarized:
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0]

The second entry in the vector is set to one (since the first entry corresponds to the label 0 , while all other entries are set to zero.

included the one-hot encoding representations for each digit, *0−9*, in the

```
0: [1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
1: [0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
2: [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
3: [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
4: [0, 0, 0, 0, 1, 0, 0, 0, 0, 0]
5: [0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
6: [0, 0, 0, 0, 0, 0, 1, 0, 0, 0]
7: [0, 0, 0, 0, 0, 0, 0, 1, 0, 0]
8: [0, 0, 0, 0, 0, 0, 0, 0, 1, 0]
9: [0, 0, 0, 0, 0, 0, 0, 0, 0, 1]
```

```
# define the 784-256-128-10 architecture using Keras
model = Sequential()
model.add(Dense(256, input_shape=(784,), activation="sigmoid"))
model.add(Dense(128, activation="sigmoid"))
model.add(Dense(10, activation="softmax"))
# train the model using SGD
print("[INFO] training network...")
sgd = SGD(0.01)
model.compile(loss="categorical_crossentropy", optimizer=sgd,
metrics=["accuracy"])
H = model.fit(trainX, trainY, validation_data=(testX, testY),
epochs=100, batch_size=128)
# evaluate the network
print("[INFO] evaluating network...")
predictions = model.predict(testX, batch_size=128)
print(classification_report(testY.argmax(axis=1),
predictions.argmax(axis=1),
target_names=[str(x) for x in lb.classes_]))
# plot the training loss and accuracy
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, 100), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, 100), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, 100), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, 100), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()
plt.savefig(args["output"])
$ python keras_mnist.py --output output/keras_mnist.png
[INFO] loading MNIST (full) dataset...
[INFO] training network...
Train on 52500 samples, validate on 17500 samples
Epoch 1/100
1s - loss: 2.2997 - acc: 0.1088 - val_loss: 2.2918 - val_acc: 0.1145
Epoch 2/100
1s - loss: 0.2792 - acc: 0.9204 - val_loss: 0.2844 - val_acc: 0.9160
[INFO] evaluating network...
precision recall f1-score support
0.0 0.94 0.96 0.95 1726
1.0 0.95 0.97 0.96 2004
2.0 0.91 0.89 0.90 1747
3.0 0.91 0.88 0.89 1828
4.0 0.91 0.93 0.92 1686

avg / total 0.92 0.92 0.92 17500
```



Conclusion:

Thus we have studied feed forward neural network architecture and its implementation using keras packages

FAQ:

1. What is neural network?

2. What is perceptron?

3. How to build model in keras?

4. What is feed forward means?

5.what is SGD?

Output:

Assignment No.3

Aim: Build the Image classification model

Objectives:

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

a. Loading and preprocessing the image data

b. Defining the model's architecture

c. Training the model

d. Estimating the model's performance

Problem statement:

More than 25% of the entire revenue in E-Commerce is attributed to apparel , accessories. A major problem they face is categorizing these apparels from just the images especially when the categories provided by the brands are inconsistent. This poses an interesting computer vision problem that has caught the eyes of several deep learning researchers.

Fashion MNIST is a drop-in replacement for the very well known, machine learning hello world – MNIST dataset which can be checked out at 'Identify the digits' practice problem. Instead of digits, the images show a type of apparel e.g. T-shirt, trousers, bag, etc.

LINK TO DOWNLOAD DATASET:

https://datahack.analyticsvidhya.com/contest/practice-problem-identify-

the-apparels/

Theory:

What is Image Classification?

Consider the below image:

You will have instantly recognized it – it's a (swanky) car. Take a step back and analyze how you came to this conclusion – you were shown an image and you classified the class it belonged to (a car, in this instance) There are potentially n number of categories in which a given image can be classified. Manually checking and classifying images is a very tedious process.

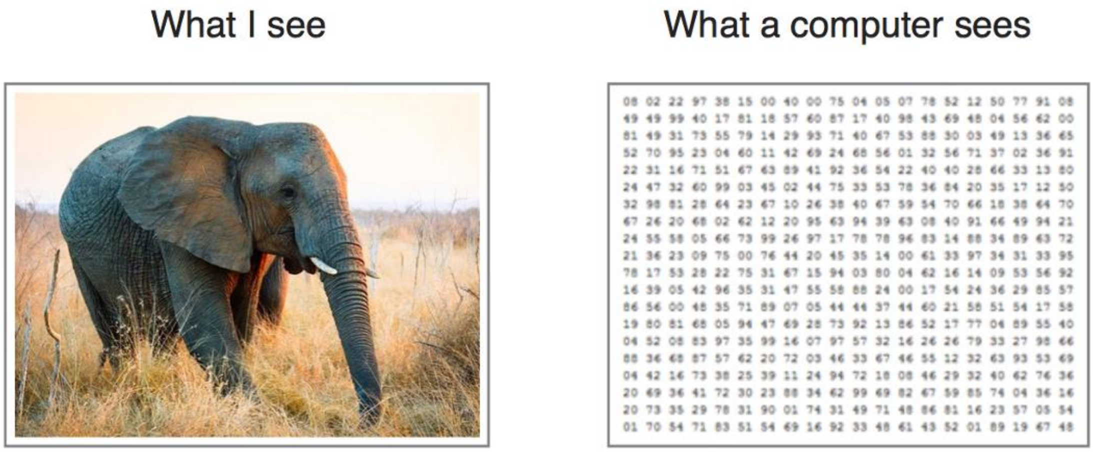
The task becomes near impossible when we're faced with a massive number of images, say 10,000 or even 100,000. How useful would it be if we could automate this entire process and quickly label images per their corresponding class?

Self-driving cars are a great example to understand where image classification is used in the real-world. To enable autonomous driving, we can build an image classification model that recognizes various objects, such as vehicles, people, moving objects, etc. on the road.

Convolutional neural networks and image classification

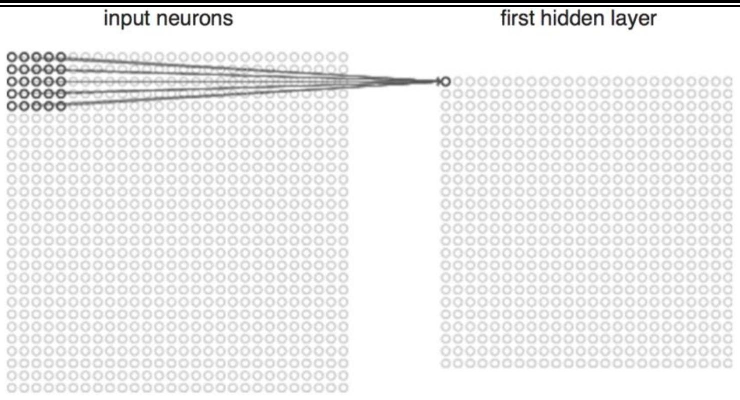
Convolutional neural networks (CNN) are a special architecture of artificial neural networks, proposed by Yann LeCun in 1988. CNN uses some features of the visual cortex. One of the most popular uses of this architecture is image classification. For example Facebook uses CNN for automatic tagging algorithms, Amazon — for generating product recommendations and Google — for search through among users' photos.

The main task of image classification is acceptance of the input image and the following definition of its class. This is a skill that people learn from their birth and are able to easily determine that the image in the picture is an elephant. But the computer sees the pictures quite differently:



Instead of the image, the computer sees an array of pixels. For example, if image size is 300 x 300. In this case, the size of the array will be 300x300x3. Where 300 is width, next 300 is height and 3 is RGB channel values. The computer is assigned a value from 0 to 255 to each of these numbers. This value describes the intensity of the pixel at each point.

The Convolution layer is always the first. The image (matrix with pixel values) is entered into it. Imagine that the reading of the input matrix begins at the top left of image. Next the software selects a smaller matrix there, which is called a **filter** (or neuron, or core). Then the filter produces convolution, i.e. moves along the input image. The filter's task is to multiply its values by the original pixel values. All these multiplications are summed up. One number is obtained in the end.



The network will consist of several convolutional networks mixed with nonlinear and pooling layers. When the image passes through one convolution layer, the output of the first layer becomes the input for the second layer. And this happens with every further convolutional layer. **The nonlinear layer** is added after each convolution operation. It has an activation function, which brings nonlinear property. The pooling layer follows the nonlinear layer. It works with width and height of the image and performs a downsampling operation on them. As a result the image volume is reduced
After completion of series of convolutional, nonlinear and pooling layers, it is necessary to attach **a fully connected layer**. This layer takes the output information from convolutional networks. Attaching a fully connected layer to the end of the network results in an N dimensional vector, where N is the amount of classes from which the model selects the desired class.

Algorithms/Pseudo codes

Setting up the Structure of our Image Data

- You should have 2 folders, one for the train set and the other for the test set. In the training set, you will have a .csv file and an image folder.
- The .csv file contains the names of all the training images and their corresponding true labels.
- The image folder has all the training images.
- The .csv file in our test set is different from the one present in the training set. This test set .csv file contains the names of all the test images.

Steps to Build our Model

- **Step 1-Create a new Python 3 notebook**
- install PyDrive. Now we will import a few required libraries:

```
 !pip install PyDrive
import os
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client  import GoogleCredentials
```

Step 2 : Import the libraries we'll need during our model building phase

```
import keras
```

```
from keras.models import Sequential
from keras.layers import Dense,Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.utils import to_categorical
from keras.preprocessing import image
import numpy as np model building phase.
import pandas as pd
 import matplotlib.pyplot as plt
 from sklearn.model_selection import train_test_split
 from keras.utils import to_categorical
from tqdm import tqdm
```

Step 3: Recall the pre-processing steps we discussed earlier. We'll be using them here after loading the data.

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

- Next, we will read all the training images, store them in a list, and finally convert that list into a numpy array.
- We have grayscale images, so while loading the images we will keep grayscale=True, if you have RGB images, you should set grayscale as False

```
train_image = []
for i in tqdm(range(train.shape[0])):
 img = image.load_
img('train/'+train['id'][i].astype('str')+'.png', target_size=(28,28,1), grayscale=True)
img = image.img_to_array(img)
 img = img/255
train_image.append(img)
 X = np.array(train_image)
```

Step 4: Creating a validation set from the training data.

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

Step 5: Define the model structure.

- This is another crucial step in our deep learning model building process. We have to define how our model will look and that requires answering questions like:

  How many convolution layers do we want?

  What should be the activation function for each layer?

  How many hidden units should each layer have?

- We will create a simple architecture with 2 convolutional layers, one dense hidden layer and an output layer.
- Next, we will compile the model we've created.

Step 6: Training the model.

- in this step, we will train the model on the training set images and validate it using, you guessed it, the validation set.

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

Step 7: Making predictions!

We'll initially follow the steps we performed when dealing with the training data. Load the test images and predict their classes using the *model.predict_classes()* function.

```
# making predictions
prediction = model.predict_classes(test)
```

# creating submission file

- sample = pd.read_csv('sample_submission_I5njJSF.csv')
- sample['label'] = prediction
- sample.to_csv('sample_cnn.csv', header=True, index=False)

Download this *sample_cnn.csv* file for you result

Conclusion:

Thus we have successfully build image classification model on Fashion MNIST Dataset.

FAQ:

1. How image classification works?

2. How we can use CNN in image classification?

3. What is Pydrive?

4. What is tqdm?

5. Explain various layers in keras, Dense,Flatten,Dropout

Output: