Deep learning

Unit - 3

Neural Networks

1. Anatomy of Neural Networks
2. Introduction to Keras
   (a) Keras
   (b) Tensorflow
3. Theano and CNTK
4. Setting up Deep Learning workstation
5. classifying Movie reviews
   (a) Binary classification
6. classifying news wires
   (a) Multi class classification

Neural Networks

(1.) Anatomy of Neural Networks :-

⇒ Anatomy :-

⇒ A Study of the Structure (or) Research about Something

⇒ Here we have to Study about the Neural Networks Completely.

**

1. what is Artificial Neural Networks ?

**

2. How we train the Neural Networks ?

**

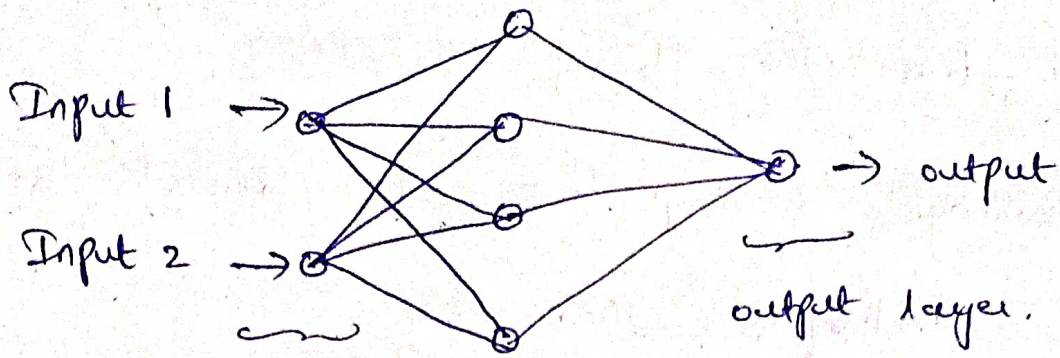
3. How to Evaluate the Performance of Neural Networks ?

1. what is Artificial Neural Networks (ANN) ?

⇒ Artificial Neural Networks (ANN) are algorithms based on brain function and are used to model Complicated patterns and forecast issues.

⇒ There are three layers in the Network Architecture

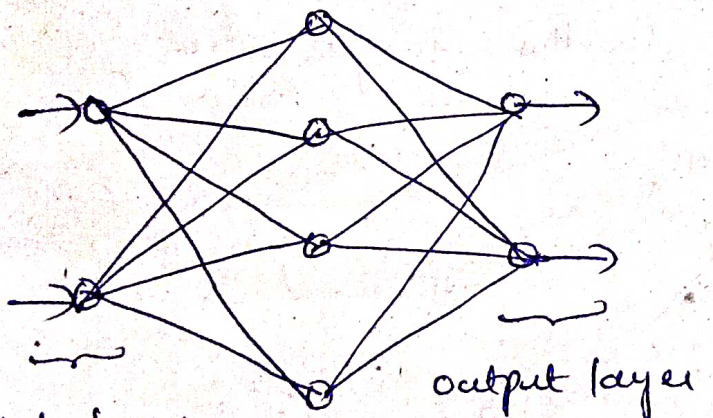
1. Input layer 2. Hidden layer 3. output layer.



⇒ In ANN Specially there are two types of Networks which we have been basically.

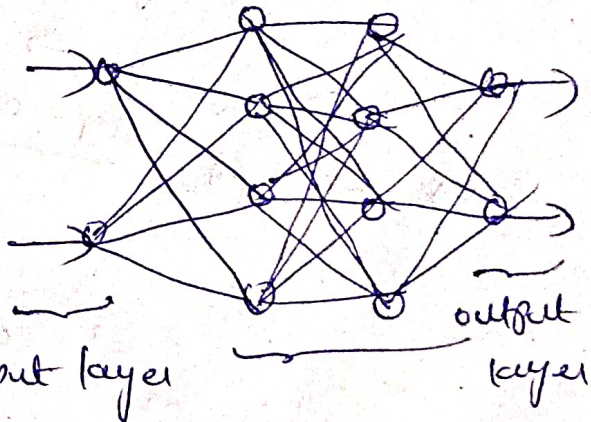
1. SNN (Single Neural Network)
2. MNN (Multi Neural Network)

1. SNN



⇒ only one hidden layer will be there for SNN

2. MNN



⇒ More than one hidden layers will be there for MNN.

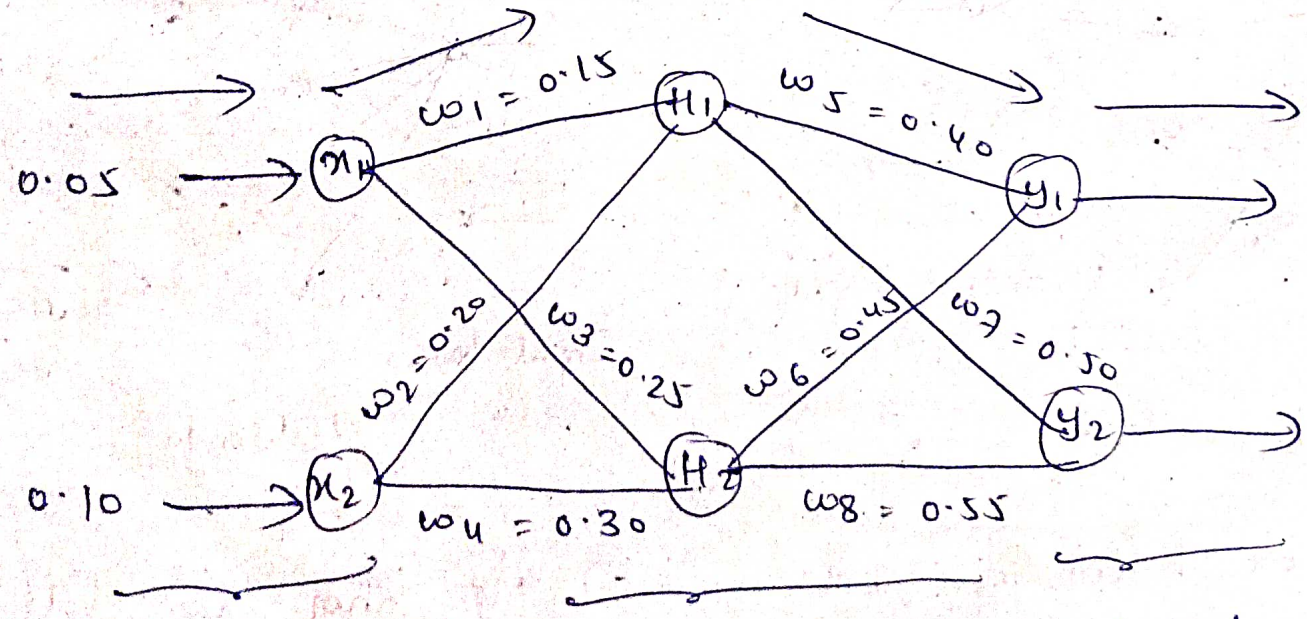
2. How we train the Neural Networks ?

=> For Training a deep Neural Networks are often use two techniques those are.

1. Forward Propagation
2. Backward Propagation.

1. Forward Propagation :-

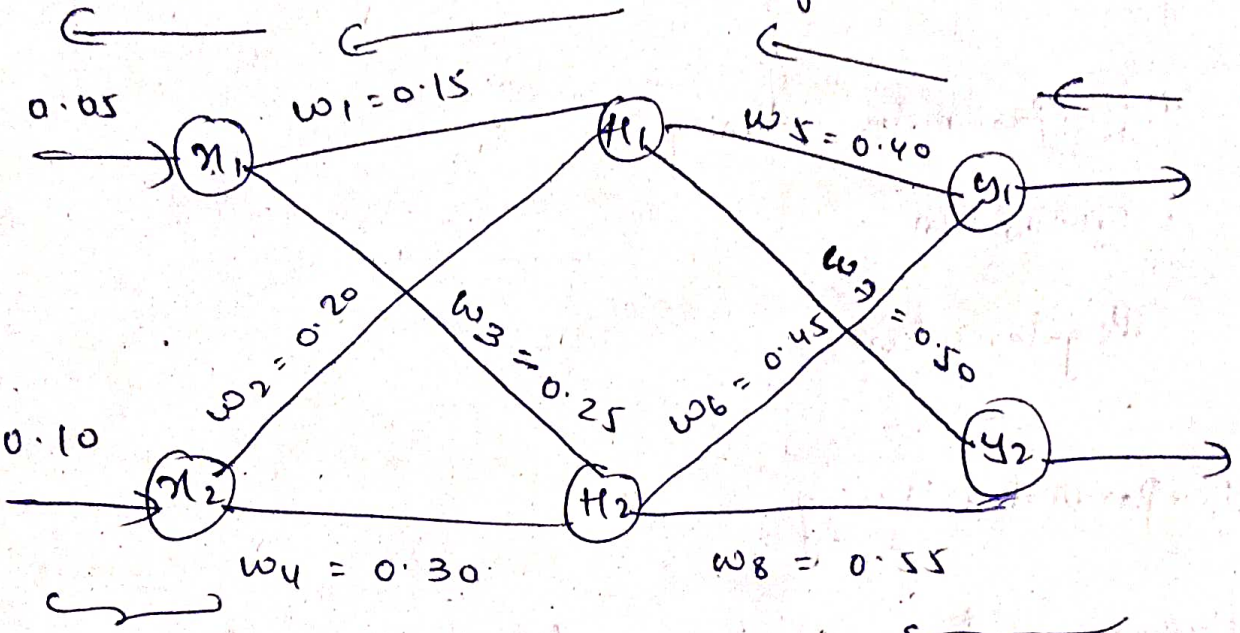
=> In the name itself Forward Propagation is used to do the process with in the flow of the layers. like, input layer, hidden layer and output layer according to the target values.



Forward Propagation

2. Backward Propagation :-

=) In Backward Propagation we have to update the weights from Backwards if the target values are not reached in forward Propagation



Input layer | Hidden layer | output layer

Backward propagation

$$H_1 = x_1 w_1 + x_2 w_2 + b_1$$

Activation function is Sigmoid $= \frac{1}{1+e^{-x}}$

output $H_1 = \frac{1}{1+e^{-H_1}}$

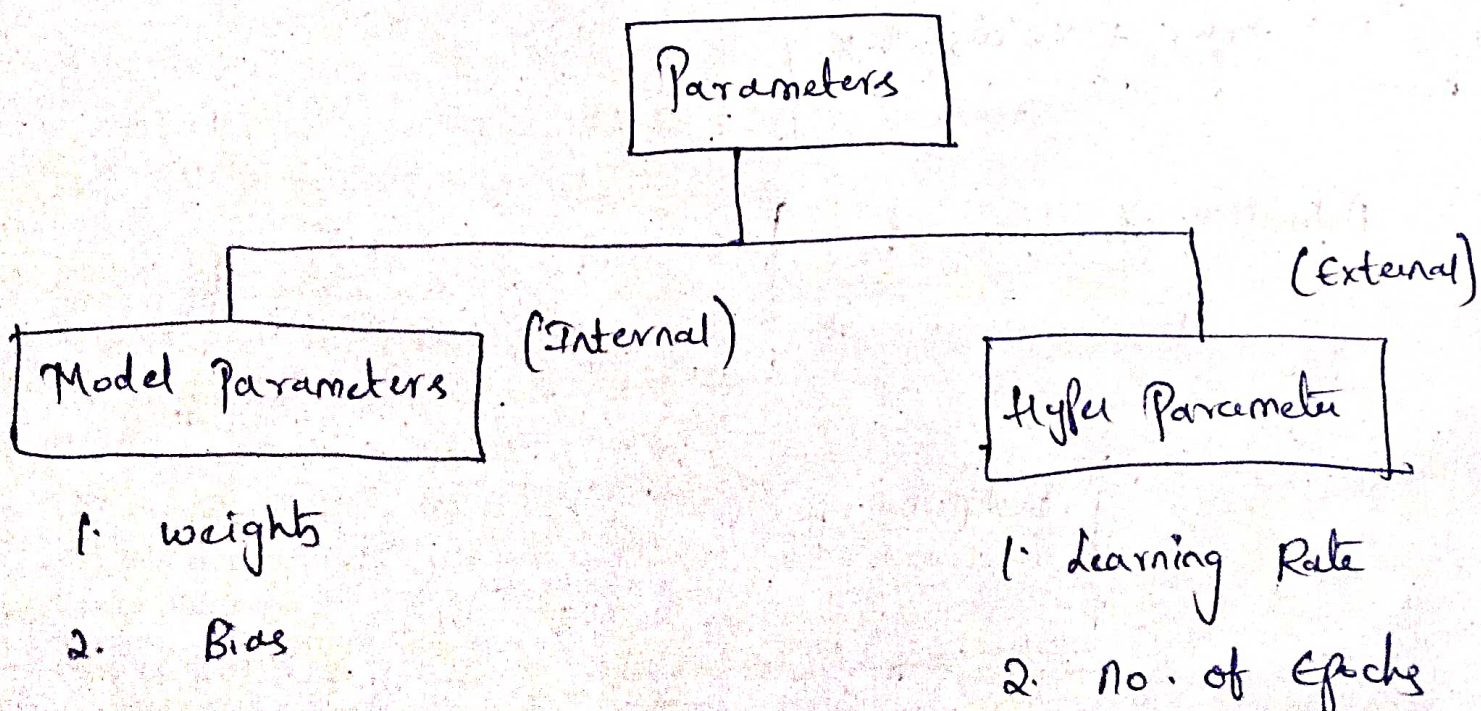
3. How to Evaluate the Performance of Neural Networks ?

⇒ To improve the Perfprmance of the deep Neural networks we have to gone through three approaches they are

1. Hyperparameter tuning
2. optimization
3. Regularization.

1. Hyperparameter Tuning :-

⇒ Hyperparameter Tuning Consists of finding a set of optimal hyperparameter Values for a learning algorithm while applying the optimized algorithm to any dataset



2. optimization techniques :-

⇒ optimization techniques algorithms are responsible for reducing losses and provide most accurate results possible.

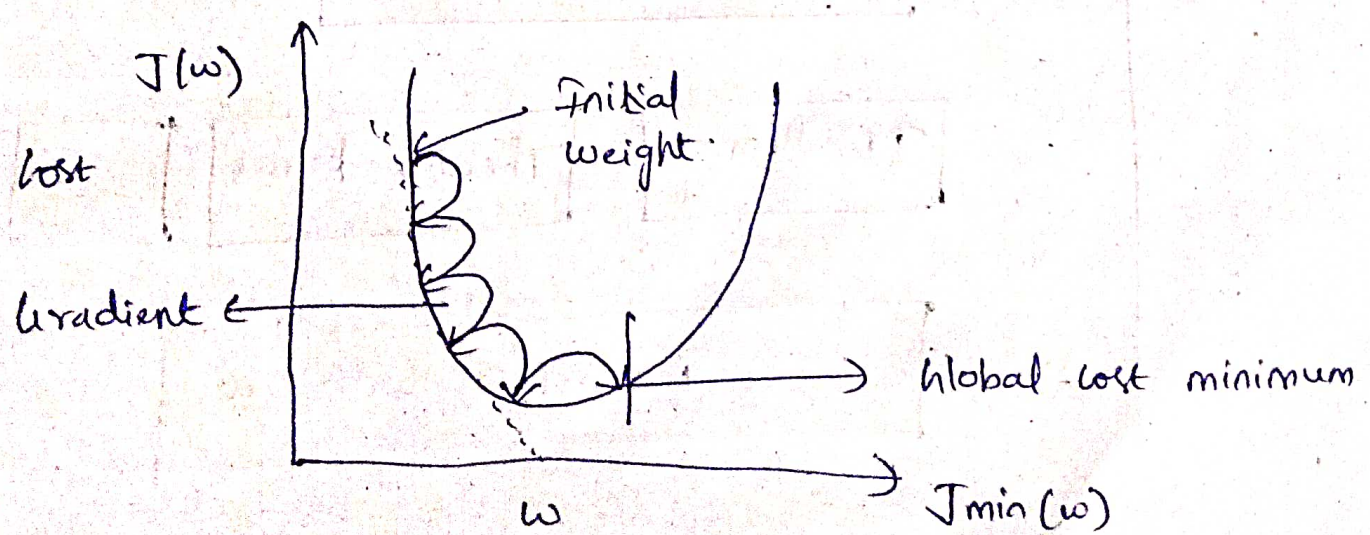
⇒ Techniques used in this optimization are :-

1. Gradient Descent
2. Stochastic Gradient Descent (SGD)
3. Mini - batch Stochastic Gradient Descent (MB-SGD)

1. Gradient Descent :-

$$\text{Cost function} = \theta_j = \theta_j - \alpha \cdot \nabla_j(\theta)$$

⇒ A Gradient Descent is an iterative algorithm, that starts from a random point on the function and traverses down its slope in steps until it reaches lowest point of that function.

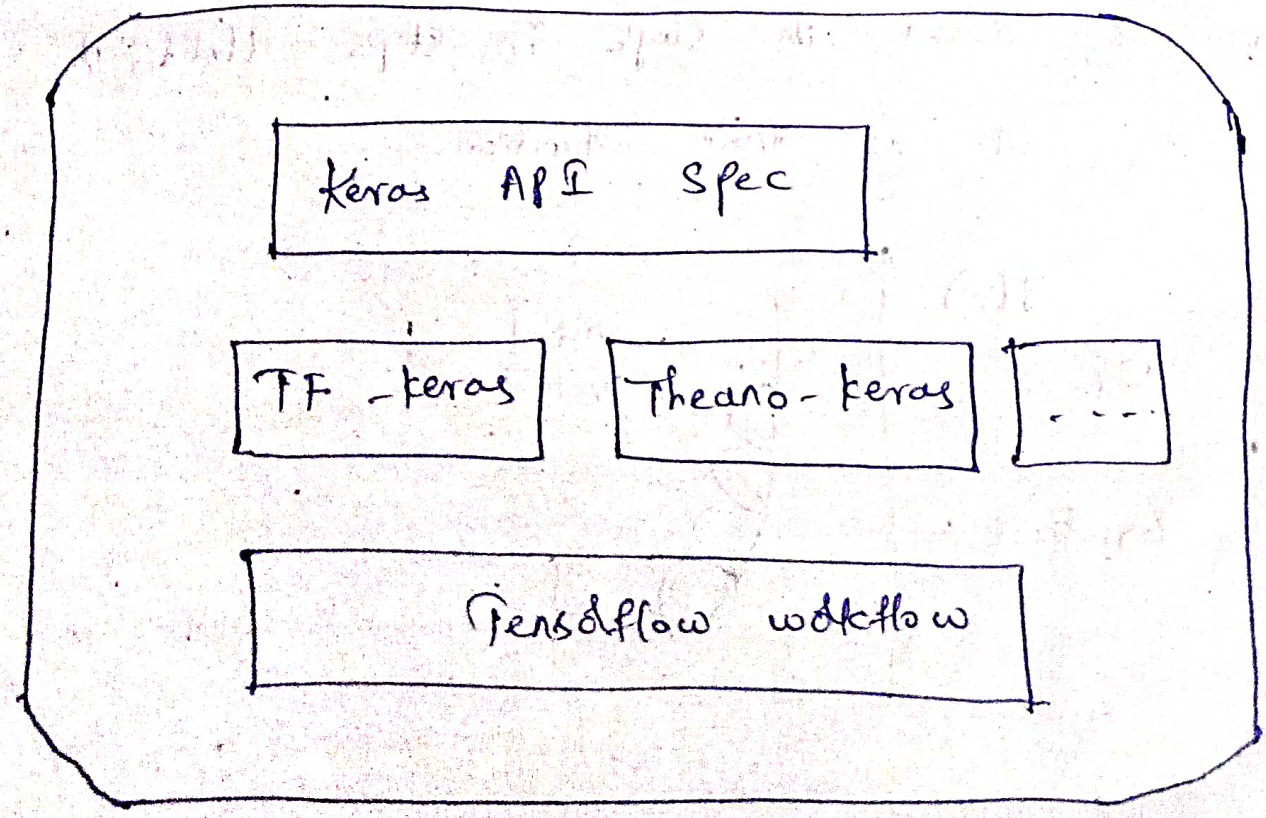


(2) Introduction to Keras :-

(a) 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 to work with because it provides a python front end with a high level of abstraction while having the option of multiple backends for computation purposes



⇒ why do we need keras ?

⇒ keras is an API that was made to be easy to learn ff people. keras was made to be simple it offers consistent & simple API's reduces the actions required to implement common code and Explains User. Errd 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.

Basic Example Using Keras Library

Following is a basic example to demonstrate how easy it is to train a model and do things like evaluation, prediction etc. Do not worry if you do not understand any of the steps described below. It is meant only for introducing development with Keras to you. We shall go in deep in our subsequent tutorials, and also through many examples to get expertise in Keras.

Import from Keras

.

Sequential() is a simple model available in Keras. It adds layers one on another sequentially, hence Sequential model. For layers we use Dense() which takes number of nodes and activation type.

```
from keras.models import Sequential
from keras.layers import Dense
```

Dataset

We shall consider a csv file as dataset. Following is a sample of it containing three observations.

```
6,148,72,35,0,33.6,0.627,50,1
1,85,66,29,0,26.6,0.351,31,0
8,183,64,0,0,23.3,0.672,32,1
```

First eight columns are features of an experiment while the last(ninth) column is output label.

You can download the dataset from here.

```
import numpy

# load dataset
dataset = numpy.loadtxt("input-data.csv", delimiter=",")
# split into input (X) and output (Y) variables
X = dataset[:,0:8]
Y = dataset[:,8]
```

In this example, we shall train a binary classifier. Output labels are either 1 or 0.

Model

Now, we define model using Keras Sequential() and Dense() classes.

```
model = Sequential()
model.add(Dense(10, input_dim=8, activation='relu'))
model.add(Dense(5, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

The code is simple and easy to read. We created a Sequential() model and added three Dense() layers to it. The first Dense layer consists of 10 nodes, each node receives input from eight input nodes and the activation used for the node is relu (rectified linear unit). The second layer has 5 nodes and the activation function used is relu. The third layer is our output node and has only one node, whose activation is sigmoid, to output 1 or 0. So, apart from input and output, we have two layers in between them. You can add some more layers in between with different activation layers. The selection has to be done by considering type of data, and can also be done

Complete Python Program – Keras Binary Classifier

Consolidating all the above steps, we get the following python program.

Python Program

```
from keras.models import Sequential
from keras.layers import Dense
import numpy
# fix random seed for reproducibility
numpy.random.seed(7)
print('random seed set')

# load dataset
dataset = numpy.loadtxt("input-data.csv", delimiter=",")
# split into input (X) and output (Y) variables
```

```
X = dataset[:,0:8]
Y = dataset[:,8]
print('input data loaded')

# create model
model = Sequential()
model.add(Dense(10, input_dim=8, activation='relu'))
model.add(Dense(5, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
print('model created')

# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print('compiled')

# Fit the model
model.fit(X, Y, epochs=150, batch_size=10)
print('data fit to model')

# evaluate the model
scores = model.evaluate(X, Y)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

Conclusion

In this Keras Tutorial, we have learnt what Keras is, its features, installation of Keras, its dependencies and how easy it is to use Keras to build a model with the help of a basic binary classifier example.

(3) Theano :-

⇒ Theano is a python library that allows us to Evaluate mathematical operations including multi-dimensional array so Efficiently.

⇒ It is mostly used in building deep learning Projects it works a more faster on GPU (Graphical Processing Unit) rather than CPU.

```
Pip install theano
```

⇒ Theano is a Python library also known as the grandfather of deep learning libraries is popular among researchers.

⇒ It is a Compiler for manupulating and analyzing mathematical Expressions, Particularly matrix valued Expressions

⇒ Computations in Theano are written in Numpy like Syntax deep learning Neural Network models to run Efficiently at high Speeds

⇒ How to Install theano ?

⇒ Anaconda in windows Prompt :-

```
Conda install -c anaconda theano
```

(or)

```
Conda install theano
```

⇒ In Command Prompt

```
Pip install theano
```

Example :- How to Add two Scalars Using Theano ?

```
# import libraries
import theano
from theano import tensor

# Creating two floating point Scalars
x = tensor.scalar()
y = tensor.scalar()
```

Creating addition Expression

z = x + y

fun = theano.function ([x, y], z)

Pass 11.6 to 'x', 1.1 to 'y'

fun (11.6, 1.1)

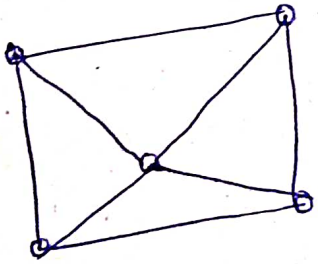
Output :-

array (12.7)

⇒ Theano library Particularly used for multi dimensional array matrices

⇒ CNTK (Microsoft Cognitive Toolkit) :-

⇒ It is a deep learning framework developed by Microsoft Research microsoft cognitive Toolkit describes neural networks as a series of computations steps via a directed graph.



→ Computational Network Toolkit

Developers :- Microsoft Research

initial release - 25th January 2016

Repository - github.com/microsoft CNTK

written in - C++, Python.

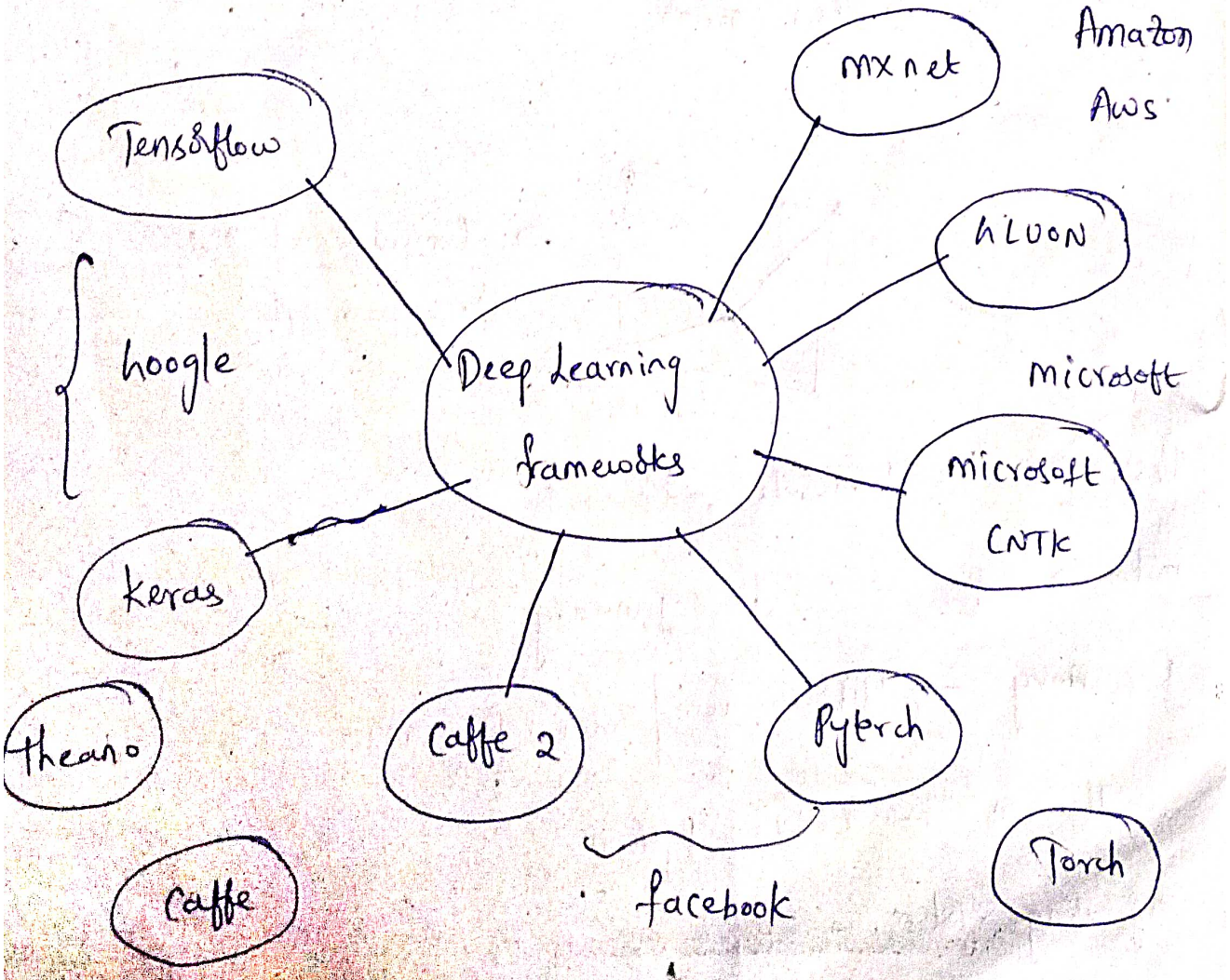
Operating system - windows, linux

Type - Library of Machine Learning & deep learning ⇒

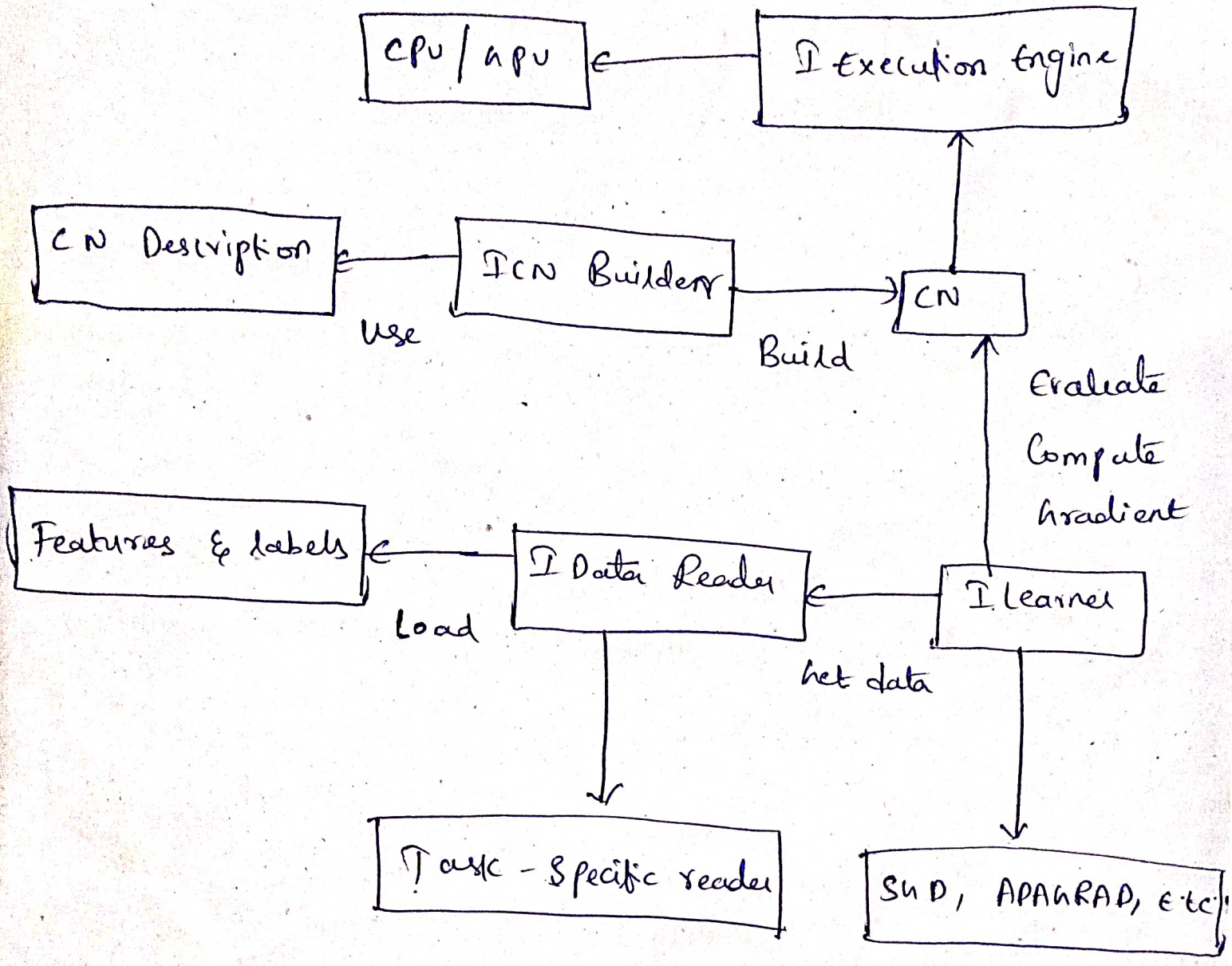
Liscense - MIT Liscense

⇒ CNTK Supports interfaces such as Python and C++ and is used for handwritting, Speech recognition and facial recognition

⇒ It Supports RNN and CNN of Neural Networks models which are capable of handling image, handwritting and Speech recognition problems.



⇒ Architecture of Microsoft Cognitive Toolkit :-



4. Setting up deep learning Workstation :-

⇒ Workstation :-

⇒ A workstation is a special computer designed for technical (or) scientific applications intended primarily to be used by a single user.

⇒ A deep learning workstation is a dedicated computer that supports compute intensive A.I and Deep learning work loads. It offers significantly higher performance compared to traditional workstations by leveraging multiple graphical processing Units (GPU's)

⇒ GPU (Graphical Processing Unit) :-

⇒ it is used to handle graphics related work like graphics, Effects and Videos.

workstation while saving $1500?

Erfan Eshratifar · Follow
Published in MLearning.ai · 3 min read · Dec 19, 2021

Companies like Lambda, Bizon, Digital Storm offer pre-built deep learning GPU rigs that are often more expensive than building the rig from scratch. However, the upside of purchase from these companies is the support and pre-built software stack you receive, but if you don't need those, building the rig from scratch saves you money. In this blog post, we aim to build the same Vector GPU workstation but $1500 cheaper! The workstation is customizable so for clarity, the following is its exact specification:

The Vector rig from Lambda labs we'd like to build from scratch. As we see, the pre-tax price of this machine is $7630 on 12/04/2021 which is $1513 higher than what we paid Newegg.com for all the pieces.

The price of each hardware piece is listed below:

Hardware	Price
Case	$195.22
Motherboard	$668.79
GPU	$1,733.96
GPU	$1,733.96
CPU	$464.99
SSD	$239.99
RAM	$599.99
Power	$379.99
Fan	$99.99
Sum	$6,116.88

Hardware piece pre-tax prices on Newegg.com purchased on 12/04/2021.

The following is the list of hardware pieces purchased with the link to the Newegg product page. The assembly instructions are also described below.

Asus 2066 Intel X299 Motherboard

- Install the power supply on the computer case. The cable management and packing of the wires are crucial here as the PSU has lots of power wires.

Asus 2066 Intel X299 Motherboard

- Install the power supply on the computer case. The cable management and packing of the wires are crucial here as the PSU has lots of power wires.

Intel Core I-9-10900X

NVIDIA GeForce RTX 3080

CORSAIR 4×32GB DDR4 RAM

- Install the motherboard on the computer case. Don't forget to attach the port plate before installing.
- Install the CPU on the motherboard. Don't touch the pins and make sure there is no dirt. Apply the thermal paste on top of the chip.
- Attach the CPU cooler to the CPU
- Install both GPUs on the motherboard

Lian Dynamic Razor Computer Case

EVGA CPU Liquid Cooler

Silicon Power 2TB SSD NVMe

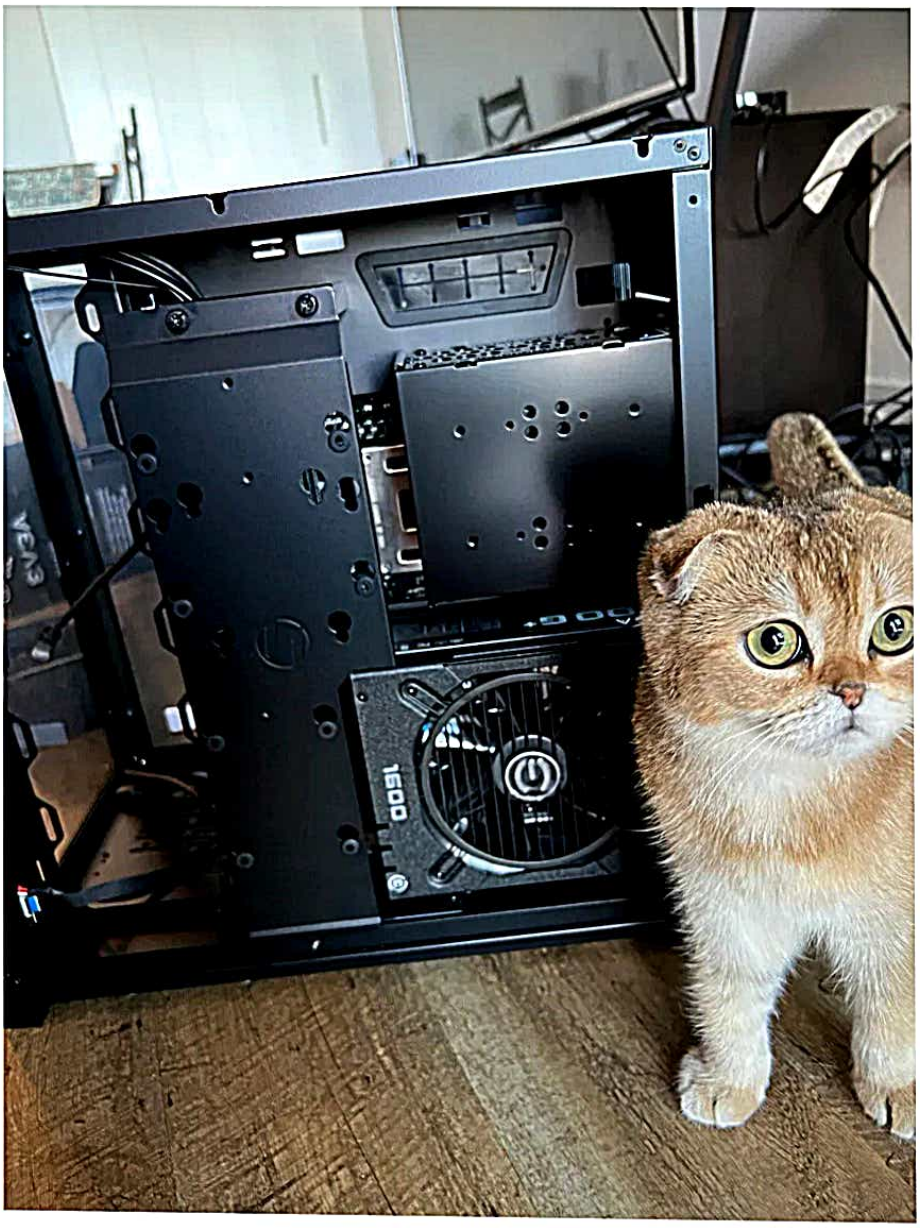
EVGA 1600W Power Supply

- Install the NVMe SSD on the motherboard
- Install all four RAMs on the motherboard
- Connect all the power cables and computer case cables such as the front power button and USB ports.

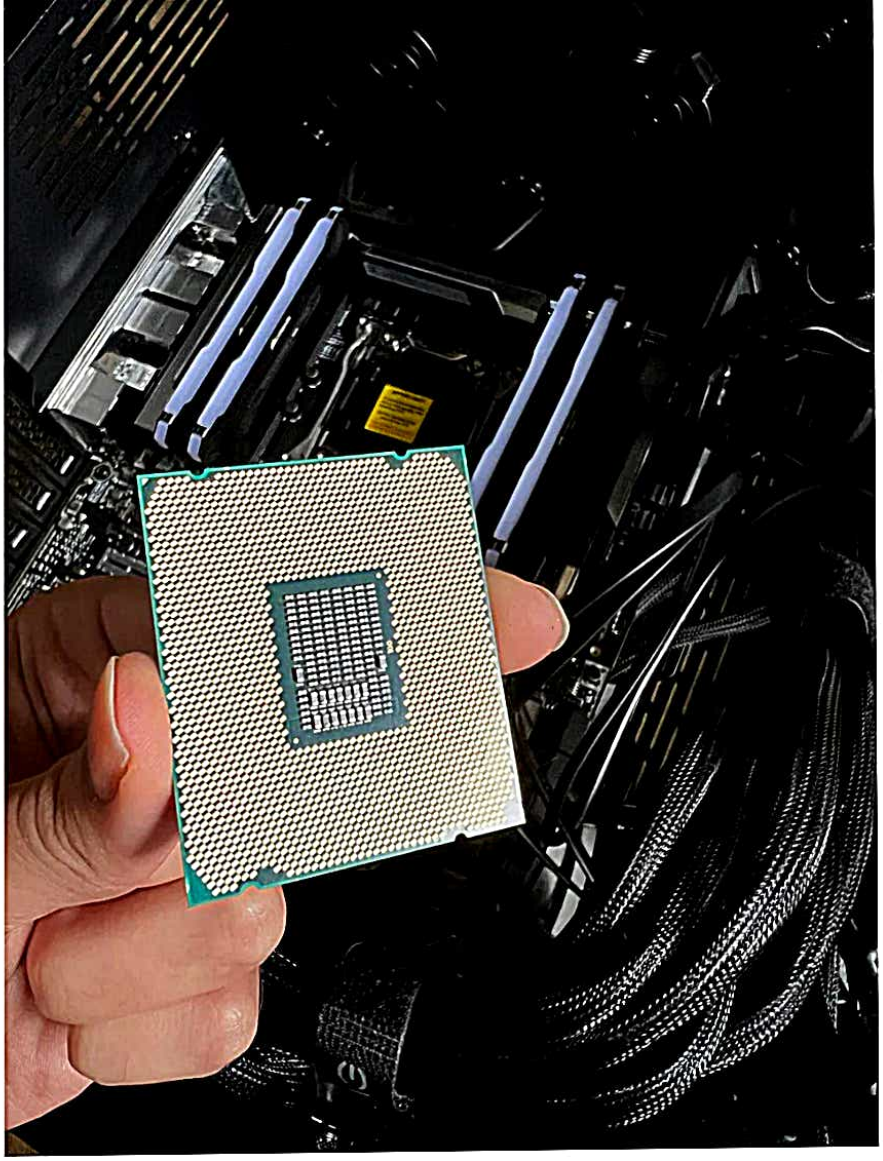
Notes:

1. If you are building a 4-GPU rig, as each GPU takes about 250–350W, you need to make sure total power is supported by your PSU and outlet.
2. Make sure the inside temperature is good by reading the temperatures of the motherboard, CPU, and GPU.

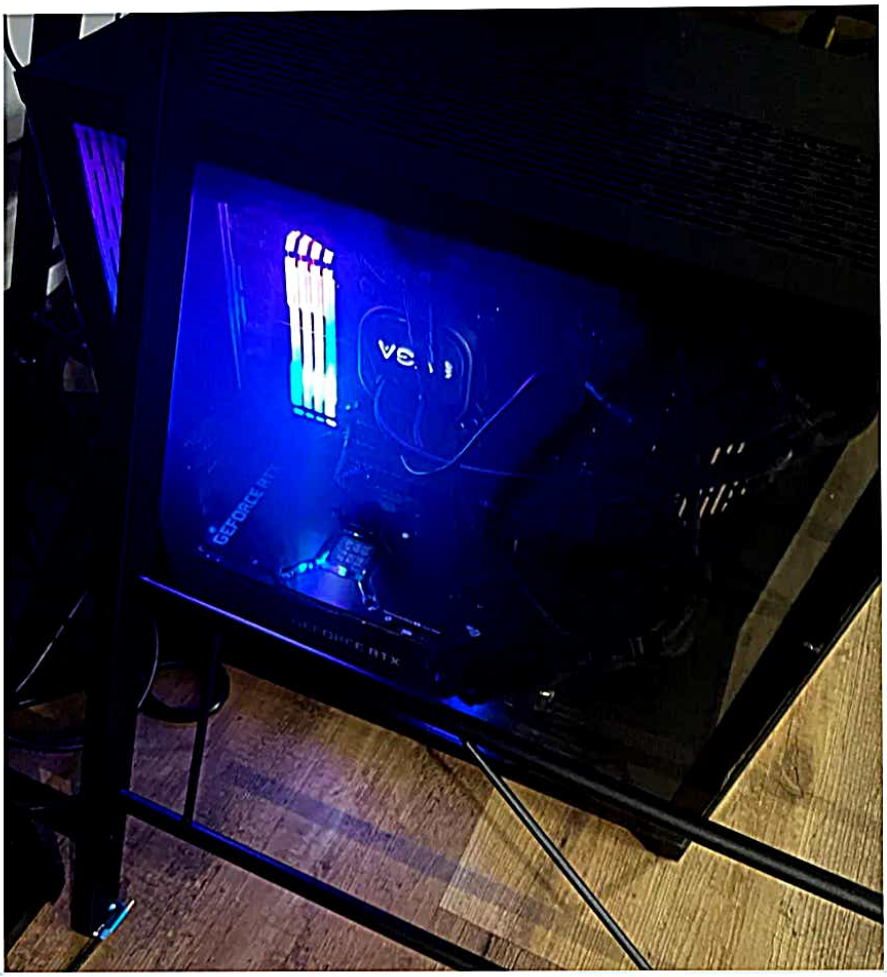
Photos during the build:



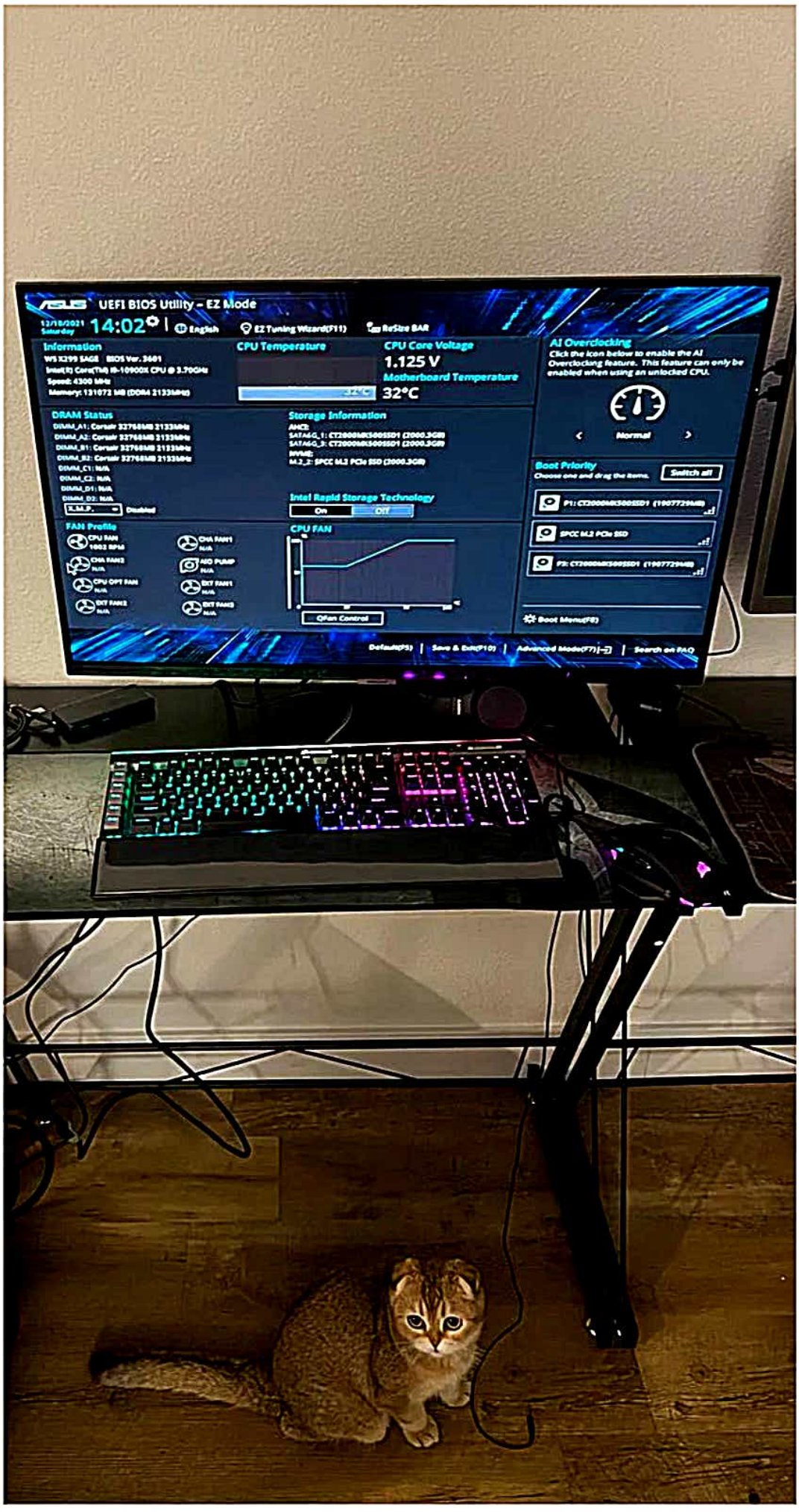
After PSU installation.



Intel Core I-9-10900X



The RAM lightning looks nice!



All hardware pieces are detected successfully!

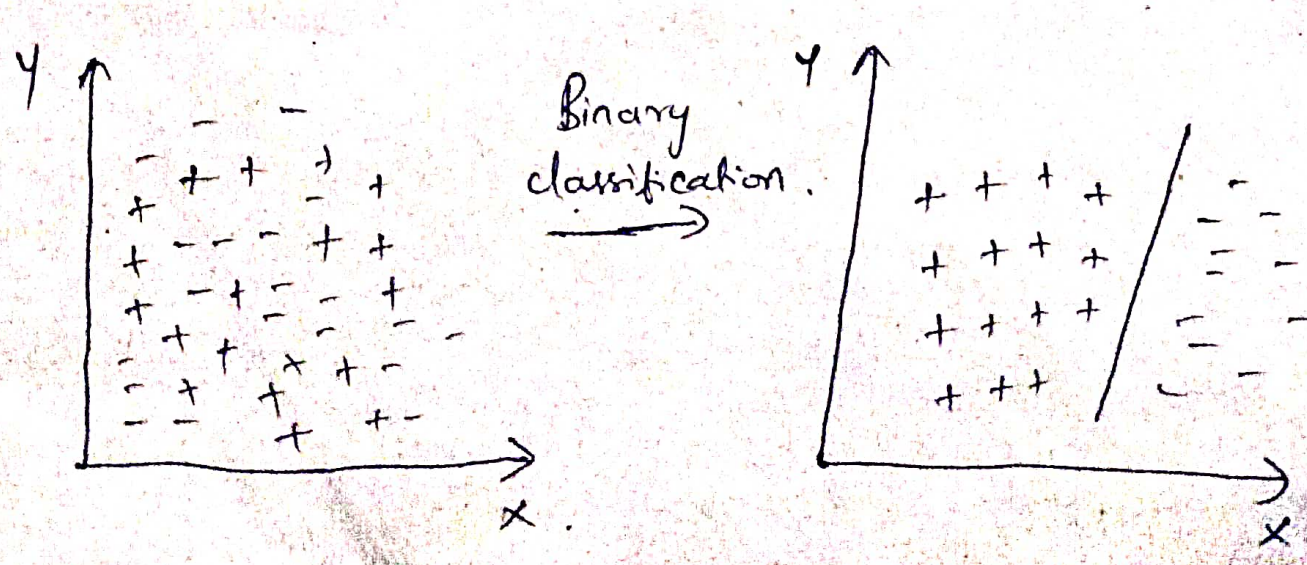
5. Classifying Movie Reviews :-

(a) Binary classification :-

⇒ Binary classification name itself knowing that we have to classify the data in to two categories, whatever the data it is we have to classify in into two ways.

⇒ we can take one case study i.e "classifying Movie Reviews" for this we will use one dataset i.e IMDB (Internet Movies Data Base) By Based up on this dataset we have to classify the Movie reviews into two categories.

1. Positive talk
2. Negative talk.



EXPERIMENT NO -2

Design a neural network for classifying movie reviews (Binary Classification) using IMDB dataset.

Procedure :

IMDB DataSet :

The IMDB (Internet Movie Database) dataset is a popular benchmark dataset for sentiment analysis, which is the task of classifying text into positive or negative categories. The dataset consists of 50,000 movie reviews, where 25,000 are used for training and 25,000 are used for testing. Each review is already preprocessed and encoded as a sequence of integers, where each integer represents a word in the review.

The goal of designing a neural network for binary classification of movie reviews using the IMDB dataset is to build a model that can classify a given movie review as either positive or negative based on the sentiment expressed in the review.

```
# Import necessary libraries
from keras.datasets import imdb
from keras.models import Sequential
from keras.layers import Dense, Dropout
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Load the dataset
```



```
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=10000)
```

In this step, we load the IMDB dataset using the imdb.load_data() function from Keras. We set the num_words parameter to 10000 to limit the number of words in each review to 10,000, which helps to reduce the dimensionality of the input data and improve model performance.

```
# Preprocess the data
maxlen = 200
X_train = pad_sequences(X_train, maxlen=maxlen)
X_test = pad_sequences(X_test, maxlen=maxlen)
```

In this step, we preprocess the data by padding the sequences with zeros to a maximum length of 200 using the pad_sequences() function from Keras. This ensures that all input sequences have the same length and can be fed into the neural network.

```
# Define the model
model = Sequential()
```

```
# Define the model
model = Sequential()
model.add(Dense(128, activation='relu', input_shape=(maxlen,)))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
```

In this step, we define the neural network architecture using the Sequential() class from Keras. Next, we define the neural network model with three fully connected layers. The first layer has 128 units with ReLU activation, the second layer has 64 units with ReLU activation, and the final layer has a single unit with sigmoid activation for binary classification.

```
# Compile the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

In this step, we compile the model using the compile() method from Keras. We

set the loss function to binary cross-entropy, which is appropriate for binary classification problems. We use the adam optimizer and track the accuracy metric during training.

```
# Train the model
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10,
batch_size=128)
```

In this step, we train the model on the training data using the fit() method from Keras. We set the number of epochs to 10 and the batch size to 128. We also pass in the test data as the validation data to monitor the performance of the model on unseen data during training.

```
# Evaluate the model on test data
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

Finally, we can evaluate the performance of the model on the test data using the evaluate() function from Keras.

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 [==============================] - 1s 0us/step
Epoch 1/10
196/196 [==============================] - 2s 7ms/step - loss: 257.8831 - accuracy: 0.4990 - val_loss:
3.1933 - val_accuracy: 0.5036
Epoch 2/10
196/196 [==============================] - 1s 6ms/step - loss: 18.6795 - accuracy: 0.5094 - val_loss:
0.7013 - val_accuracy: 0.5010
Epoch 3/10
196/196 [==============================] - 1s 6ms/step - loss: 4.7280 - accuracy: 0.4982 - val_loss:
0.6967 - val_accuracy: 0.4960
Epoch 4/10
196/196 [==============================] - 1s 6ms/step - loss: 2.4193 - accuracy: 0.5012 - val_loss:
0.6952 - val_accuracy: 0.4974
Epoch 5/10
196/196 [==============================] - 1s 5ms/step - loss: 1.5178 - accuracy: 0.5043 - val_loss:
0.6936 - val_accuracy: 0.4989
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 [==============================] - 1s 0us/step
Epoch 1/10
196/196 [==============================] - 2s 7ms/step - loss: 257.8831 - accuracy: 0.4990 - val_loss: 3.1933 - val_accuracy: 0.5036
Epoch 2/10
196/196 [==============================] - 1s 6ms/step - loss: 18.6795 - accuracy: 0.5094 - val_loss: 0.7013 - val_accuracy: 0.5010
Epoch 3/10
196/196 [==============================] - 1s 6ms/step - loss: 4.7280 - accuracy: 0.4982 - val_loss: 0.6967 - val_accuracy: 0.4960
Epoch 4/10
196/196 [==============================] - 1s 6ms/step - loss: 2.4193 - accuracy: 0.5012 - val_loss: 0.6952 - val_accuracy: 0.4974
Epoch 5/10
196/196 [==============================] - 1s 5ms/step - loss: 1.5178 - accuracy: 0.5043 - val_loss: 0.6936 - val_accuracy: 0.4989
Epoch 6/10
196/196 [==============================] - 1s 6ms/step - loss: 1.2867 - accuracy: 0.5026 - val_loss:
```



```
0.6937 - val_accuracy: 0.5003
Epoch 7/10
196/196 [==============================] - 1s 6ms/step - loss: 1.1111 - accuracy: 0.5014 - val_loss: 0.6932 - val_accuracy: 0.5002
Epoch 8/10
196/196 [==============================] - 2s 8ms/step - loss: 1.0110 - accuracy: 0.4982 - val_loss: 0.6932 - val_accuracy: 0.5002
Epoch 9/10
196/196 [==============================] - 1s 7ms/step - loss: 0.8932 - accuracy: 0.4972 - val_loss: 0.6932 - val_accuracy: 0.5004
Epoch 10/10
196/196 [==============================] - 1s 6ms/step - loss: 0.8888 - accuracy: 0.4971 - val_loss: 0.6931 - val_accuracy: 0.5002

    Accuracy: 50.02%
```

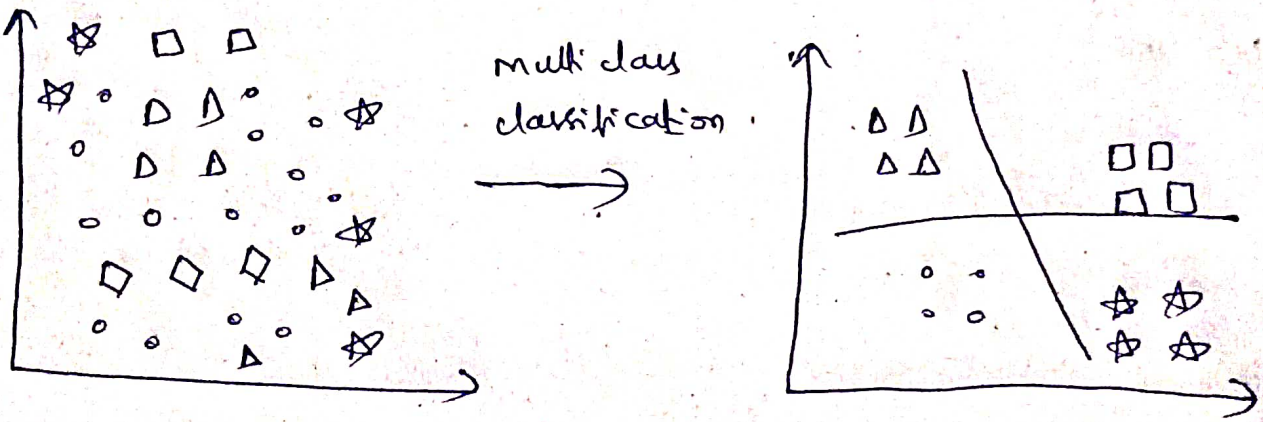
6. classifying news wires :-

(a) Multi class classification :-

⇒ Multi class classification is the problem of classifying the data into one of three (or) more classes

⇒ where we can consider any data and classify that into more than three classes

⇒ Example :-

⇒ 

⇒ where in the above example we had divide the data into 4 categories by using multi class classification.

EXPERIMENT NO -3

Design a neural Network for classifying news wires (Multi class classification) using Reuters dataset

Procedure :

Reuters DataSet :

The Reuters dataset is a collection of newswire articles and their categories. It consists of 11,228 newswire articles that are classified into 46 different topics or categories. The goal of this task is to train a neural network to accurately classify newswire articles into their respective categories.

Input layer: This layer will take in the vectorized representation of the news articles in the Reuters dataset.

Hidden layers: You can use one or more hidden layers with varying number of neurons in each layer. You can experiment with the number of layers and neurons to find the optimal configuration for your specific problem.

Output layer: This layer will output a probability distribution over the possible categories for each input news article. Since this is a multi-class classification problem, you can use a softmax activation function in the output layer to ensure that the predicted probabilities sum to 1.

```
import numpy as np
from tensorflow.keras.datasets import reuters
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.utils import to_categorical
```

We will import all the necessary libraries for the model and We will use the Keras

library to load the dataset and preprocess it.

```
# Load the Reuters dataset
(x_train, y_train), (x_test, y_test) = reuters.load_data(num_words=10000)
```

The first step is to load the Reuters dataset and preprocess it for training. We will also split the dataset into train and test sets.

In this step, we load the IMDB dataset using the reuters.load_data() function from Keras. We set the num_words parameter to 10000 to limit the number of words in each review to 10,000, which helps to reduce the dimensionality of the input data and improve model performance.

```
# Vectorize the data using one-hot encoding
def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
      results[i, sequence] = 1
    return results

x_train = vectorize_sequences(x_train)
x_test = vectorize_sequences(x_test)
```

```
x_train = vectorize_sequences(x_train)
x_test = vectorize_sequences(x_test)

# Convert the labels to one-hot vectors
num_classes = max(y_train) + 1
y_train = to_categorical(y_train, num_classes)
y_test = to_categorical(y_test, num_classes)

# Define the neural network architecture
model = Sequential()
model.add(Dense(64, activation='relu', input_shape=(10000,)))
```



```
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
```

The next step is to design the neural network architecture. For this task, we will use a fully connected neural network with an input layer, multiple hidden layers, and an output layer. We will use the Dense class in Keras to add the layers to our model. Since we have 46 categories, the output layer will have 46 neurons, and we will use the softmax activation function to ensure that the output of the model represents a probability distribution over the 46 categories.

```
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
```

Once we have defined the model architecture, the next step is to compile the model. We need to specify the loss function, optimizer, and evaluation metrics for the model. Since this is a multi-class classification problem, we will use the categorical_crossentropy loss function. We will use the adam optimizer and accuracy as the evaluation metric.

```
# Train the model on the training set
history = model.fit(x_train, y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_test, y_test))
```

After compiling the model, the next step is to train it on the training data. We will use the fit method in Keras to train the model. We will also specify the validation data and the batch size.

```
# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(x_test, y_test)
print('Test accuracy:', test_acc)
```

Evaluate the performance of the neural network on the validation set and tune the hyperparameters such as learning rate, number of layers, number of neurons, etc., based on the validation performance.

```
# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(x_test, y_test)
print('Test accuracy:', test_acc)
```

Evaluate the performance of the neural network on the validation set and tune the hyperparameters such as learning rate, number of layers, number of neurons, etc., based on the validation performance.


```
Epoch 1/20
18/18 [==============================] - 2s 53ms/step - loss: 3.5441 - accuracy: 0.1880 - val_loss: 2.9635
- val_accuracy: 0.4809
Epoch 2/20
18/18 [==============================] - 1s 38ms/step - loss: 2.6812 - accuracy: 0.4036 - val_loss: 1.9865
- val_accuracy: 0.5610
Epoch 3/20
18/18 [==============================] - 1s 44ms/step - loss: 2.0019 - accuracy: 0.5345 - val_loss: 1.6289
- val_accuracy: 0.6394
Epoch 4/20
18/18 [==============================] - 1s 38ms/step - loss: 1.6804 - accuracy: 0.6021 - val_loss: 1.4734
- val_accuracy: 0.6772
Epoch 5/20
18/18 [==============================] - 1s 39ms/step - loss: 1.4998 - accuracy: 0.6431 - val_loss: 1.3653
- val_accuracy: 0.6874
Epoch 6/20
18/18 [==============================] - 1s 39ms/step - loss: 1.3755 - accuracy: 0.6711 - val_loss: 1.2938
- val_accuracy: 0.6968
Epoch 7/20
18/18 [==============================] - 1s 44ms/step - loss: 1.2737 - accuracy: 0.6915 - val_loss: 1.2457
- val_accuracy: 0.7070
Epoch 8/20
18/18 [==============================] - 1s 38ms/step - loss: 1.1878 - accuracy: 0.7150 - val_loss: 1.2033
- val_accuracy: 0.7168
Epoch 9/20
18/18 [==============================] - 1s 66ms/step - loss: 1.0933 - accuracy: 0.7280 - val_loss: 1.1682
- val_accuracy: 0.7320
Epoch 10/20
18/18 [==============================] - 1s 61ms/step - loss: 1.0417 - accuracy: 0.7455 - val_loss: 1.1414
- val_accuracy: 0.7480
Epoch 11/20
18/18 [==============================] - 1s 38ms/step - loss: 0.9840 - accuracy: 0.7562 - val_loss: 1.1134
- val_accuracy: 0.7547
Epoch 12/20
18/18 [==============================] - 1s 38ms/step - loss: 0.9252 - accuracy: 0.7701 - val_loss: 1.0935
- val_accuracy: 0.7631
Epoch 13/20
18/18 [==============================] - 1s 38ms/step - loss: 0.8799 - accuracy: 0.7859 - val_loss: 1.0739
- val_accuracy: 0.7694
Epoch 14/20
18/18 [==============================] - 1s 38ms/step - loss: 0.8282 - accuracy: 0.7931 - val_loss: 1.0629
- val_accuracy: 0.7703
Epoch 15/20
18/18 [==============================] - 1s 42ms/step - loss: 0.7916 - accuracy: 0.7973 - val_loss: 1.0557
- val_accuracy: 0.7720
Epoch 16/20
18/18 [==============================] - 1s 38ms/step - loss: 0.7636 - accuracy: 0.8055 - val_loss: 1.0515
- val_accuracy: 0.7787
Epoch 17/20
18/18 [==============================] - 1s 38ms/step - loss: 0.7293 - accuracy: 0.8127 - val_loss: 1.0626
- val_accuracy: 0.7743
Epoch 18/20
18/18 [==============================] - 1s 38ms/step - loss: 0.7039 - accuracy: 0.8233 - val_loss: 1.0591
- val_accuracy: 0.7765
Epoch 19/20
18/18 [==============================] - 1s 39ms/step - loss: 0.6743 - accuracy: 0.8253 - val_loss: 1.0495
- val_accuracy: 0.7809
Epoch 20/20
18/18 [==============================] - 1s 38ms/step - loss: 0.6579 - accuracy: 0.8272 - val_loss: 1.0723
- val_accuracy: 0.7769
71/71 [==============================] - 0s 4ms/step - loss: 1.0723 - accuracy: 0.7769
```



```
Test accuracy: 0.7769367694854736
```