Deep Learning

Unit - 4

Convolutional Neural Networks :-

1. Neural Network and Representation learning
2. Convolutional layers
3. Multichannel Convolutional operation

Recurrent Neural Networks :-

1. Introduction to RNN
2. RNN code
3. Pytorch Tensors

(a) Deep learning with PyTorch

(b) CNN in PyTorch

⇒ Convolutional Neural Networks (CNN) :-

1. Neural Network and Representation learning :-

⇒ Neural Network converts data in such a form that it would be better to solve the desired problem. this is called representation learning.

⇒ Methods of Representation learning :-

⇒ we must employ representation learning to ensure that the model provides invarient and Untangled outcomes in order to increase its accuracy and performance.

(a) Supervised learning :-

⇒ This is referred to as Supervised learning when the ML (or) DL model maps the input X to the output Y. The computer tries to correct itself by comparing model output to ground truth. it will be using labelled input data.

Ex :- Supervised neural networks, Multilayer perceptrons etc

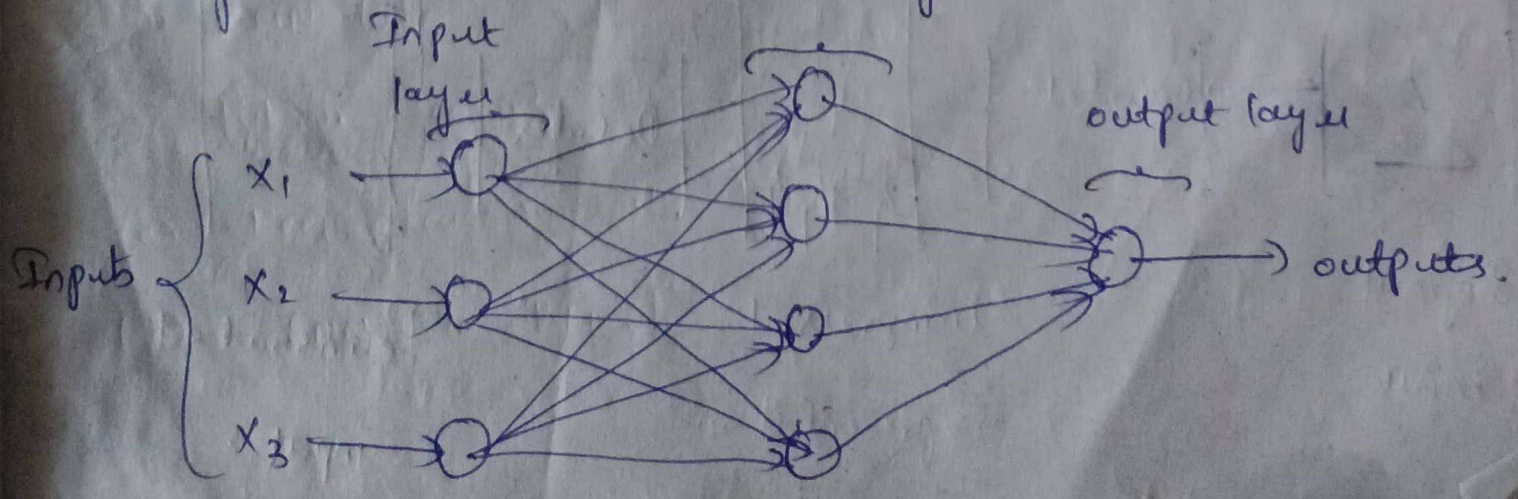
(b) Unsupervised Learning :-

⇒ Unsupervised feature learning learns features from Unlabeled input data by learning from past Experiences to Predict the feature. in this we supposed to perform clustering and association.

Ex :- Auto Encoders, matrix factorization.

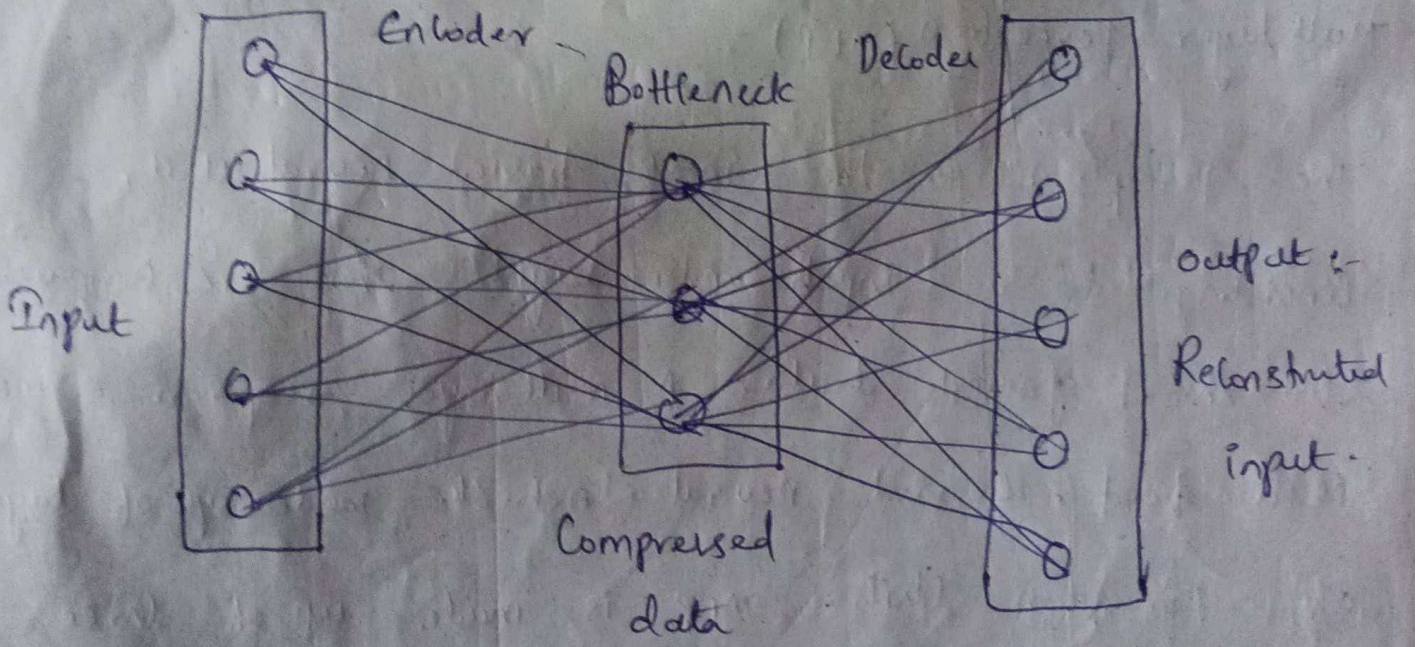
⇒ Multilayer Perceptron (MLP) :-

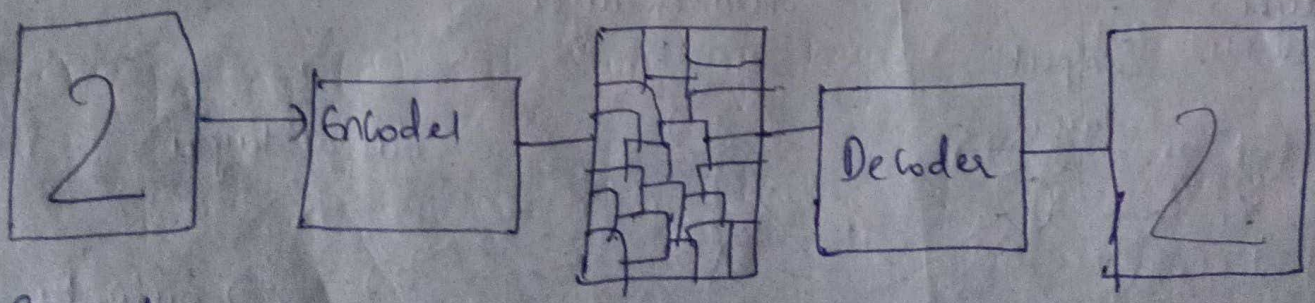
⇒ The perceptron is the most basic neural Unit, Consisting of a inputs and weights that are compared to the ground truth. A multilayer perceptron & MLP is a feed-forward neural network made up of layers of Perceptron Units. MLP is made up of three-node layers ① Input layer ② Hidden layer ③ output layer.



AutoEncoders :-

⇒ Deep learning autoEncoders are therefore neural networks that may be taught to do representation learning. AutoEncoders seek to duplicate their input to their output using an Encoder and a decoder.





(2) Convolutional Layers :-

⇒ In deep Learning, a Convolutional Neural network (CNN (or) ConvNet) is a class of deep Neural networks, that are typically used to recognize patterns present in images but they are also used for Spatial data analysis, Computer Vision, natural language Processing, Signal Processing.

⇒ CNN's make use of filters (also known as Kernels) to detect what features, such as Edges, are present throughout an image. There are four main operations in a CNN.

(a) Convolution

(b) Non-Linearity (ReLU)

(c) Pooling (or) Sub Sampling

(d) classification (fully Connected layer)

(a) Convolution layer :-

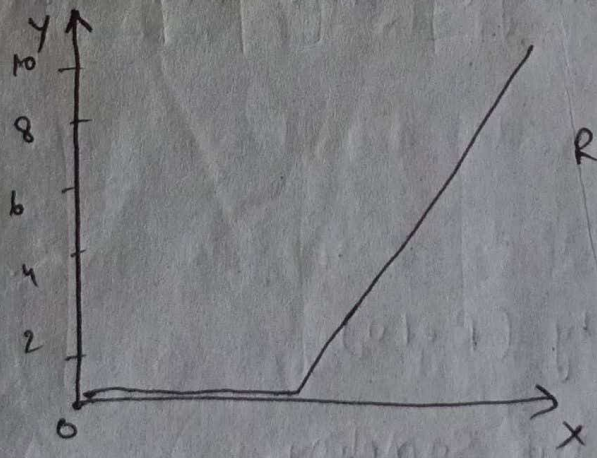
⇒ The Convolutional layer is the core building block of a CNN, and it is where the Majority Computation occurs.

it requires a few components, which are input data, a filter, and a feature map. Lets assume that the input will be a color image, which is made up of a matrix of Pixels in 3D.

(b) Non-Linearity (ReLu) :-

⇒ ReLu → Rectified Linear Unit

⇒ Relu function is another non-linear activation function main advantage of Using ReLu function over other activation functions is that it does not activate all neurons at the same time.



$R(z) = \max(0, z)$

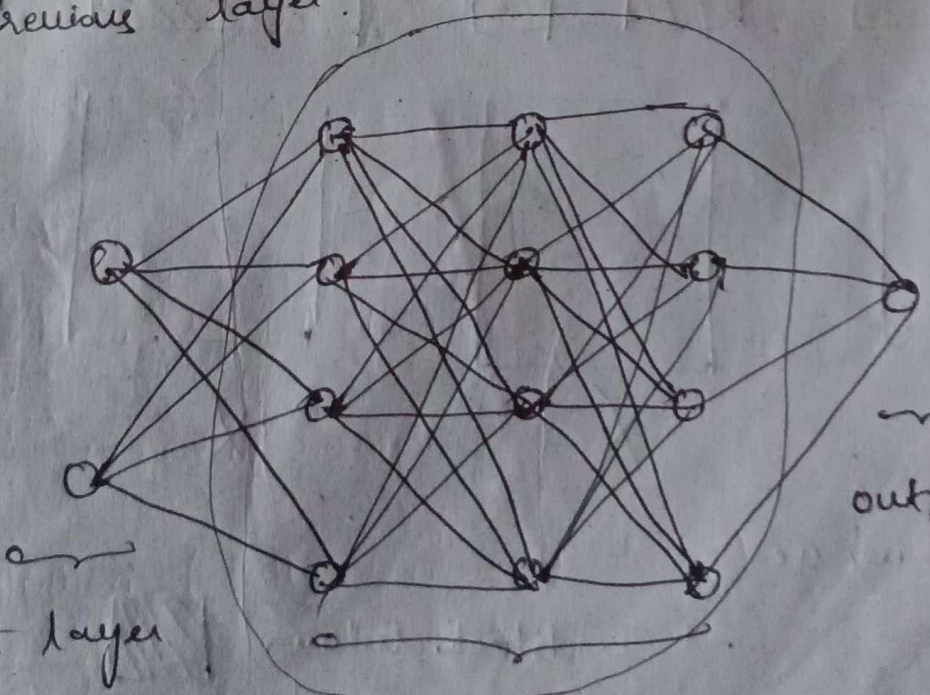
(c) Pooling (or) Sub Sampling Layer :-

⇒ Pooling layers, also known as down Sampling, Conducts dimensionality reduction, reducing the number of parameters in the input. Similar to the Convolutional layer, the Pooling

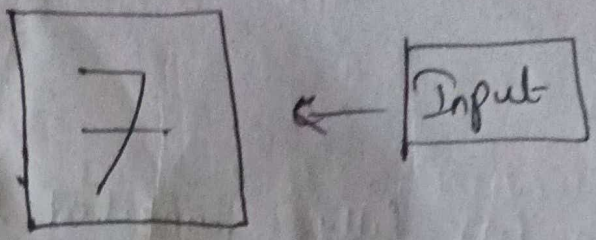
operation sweeps a filter across the entire input, but the difference is that this filter does not have any weights.

(d) classification (or) fully - connected layer :-

⇒ The name of the full - connected layer aptly describes itself. As mentioned earlier, the pixel values of the input image are not directly connected to the output layer in Partially connected layers. However, in the fully- connected layer, each node in the output layer connects directly to a node in the previous layer.



Example :- Edge → Picture representation.



Filter 1

-1	-1	-1
1	1	1
0	0	0

Filter 2

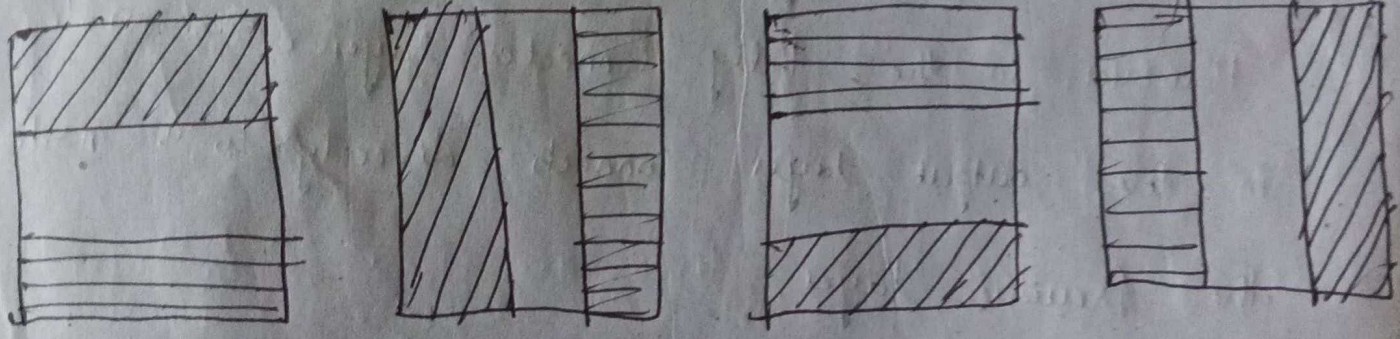
-1	1	0
-1	1	0
-1	1	0

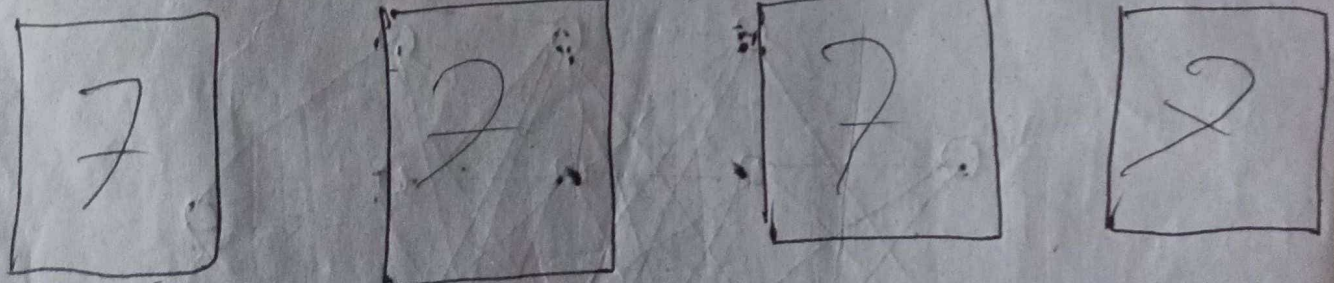
Filter 3

0	0	0
1	1	1
-1	-1	-1

Filter 4

0	1	-1
0	1	-1
0	1	-1





✓ X X X

↑ output

where

-1 = Black Colour

1 = white Colour

0 = grey Colour

(3) Multi-channel Convolutional Operation :-

⇒ So far in this Convolutional Series we have been applying the filters and arrays concepts like

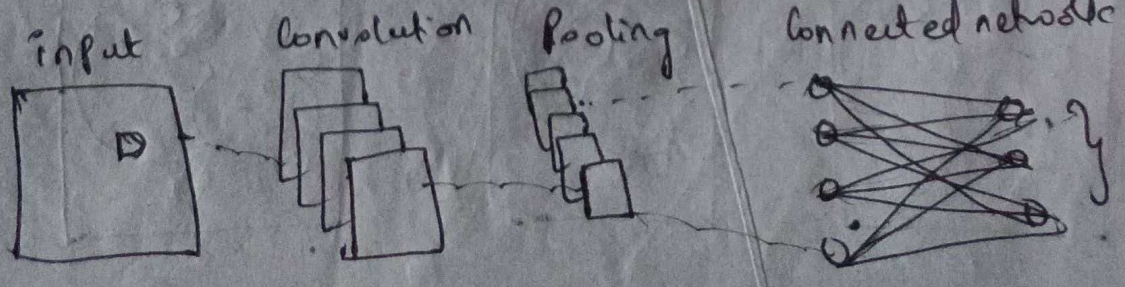
1. 1D Convolutions to 1 dimensional data (temporal)
2. 2D Convolutions to 2 dimensional data (height and width)
3. 3D Convolutions to 3 dimensional data (height, width and depth)

⇒ Our input data usually defines Multiple Variables at each Position (through time, (&) Space) and not just a single value. We call these channels.

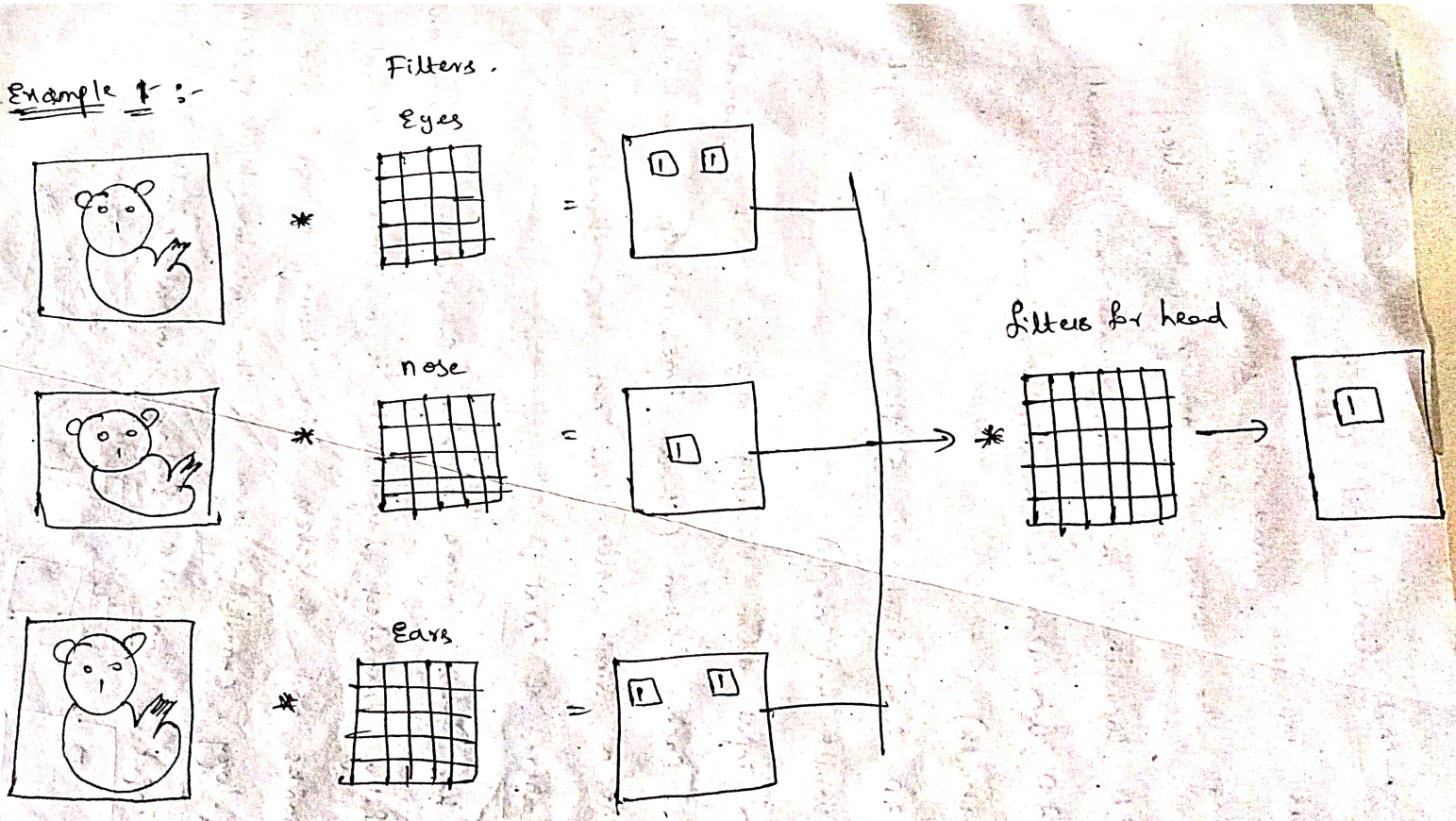
⇒ Architecture of CNN :-

⇒ It has three layers namely, Convolutional, Pooling, and a fully Connected layer. It is a class of neural networks and Processes data having a grid-like topology. Convolution layer is the building block of CNN

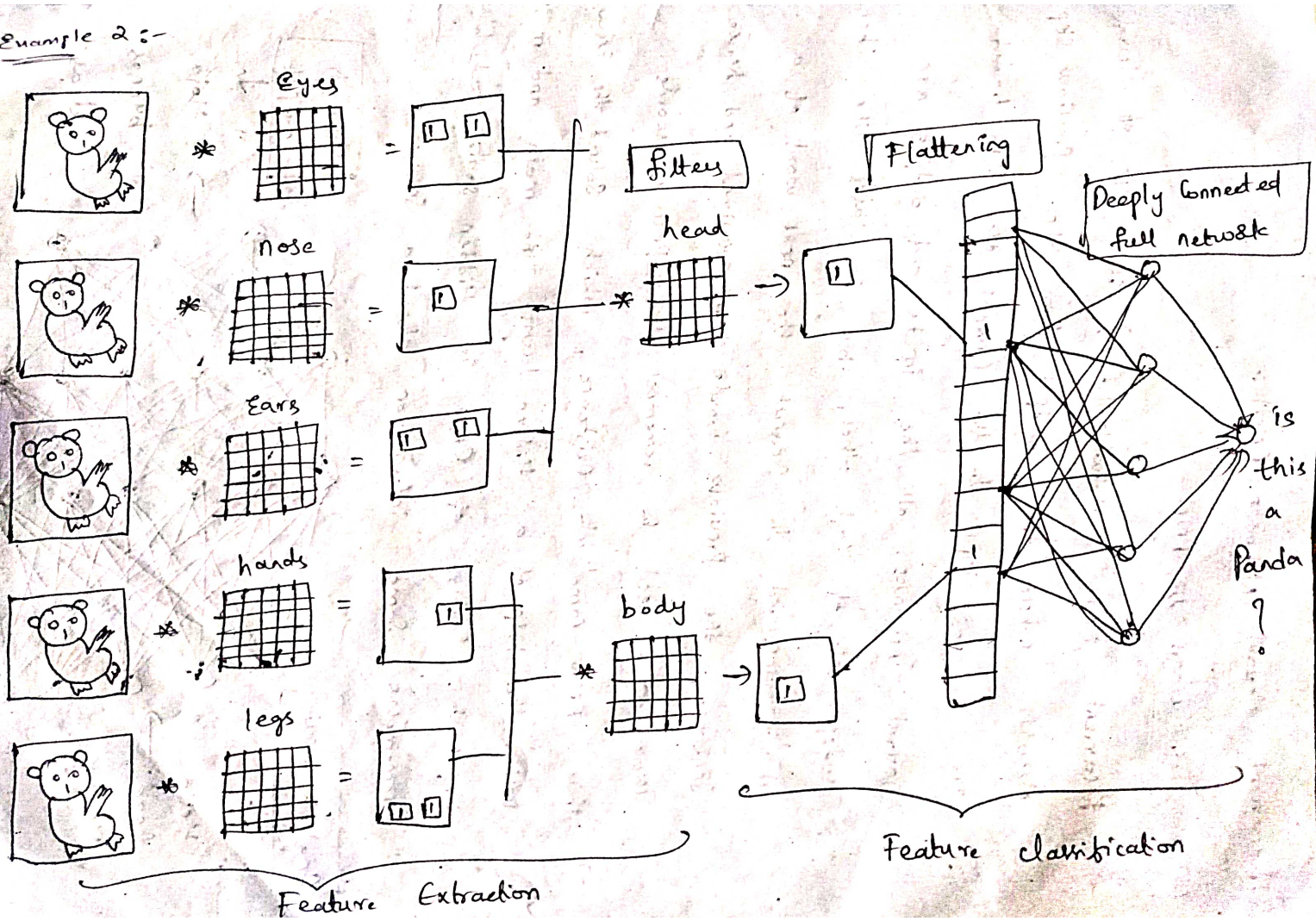
⇒



Example 1 :-



Example 2 :-

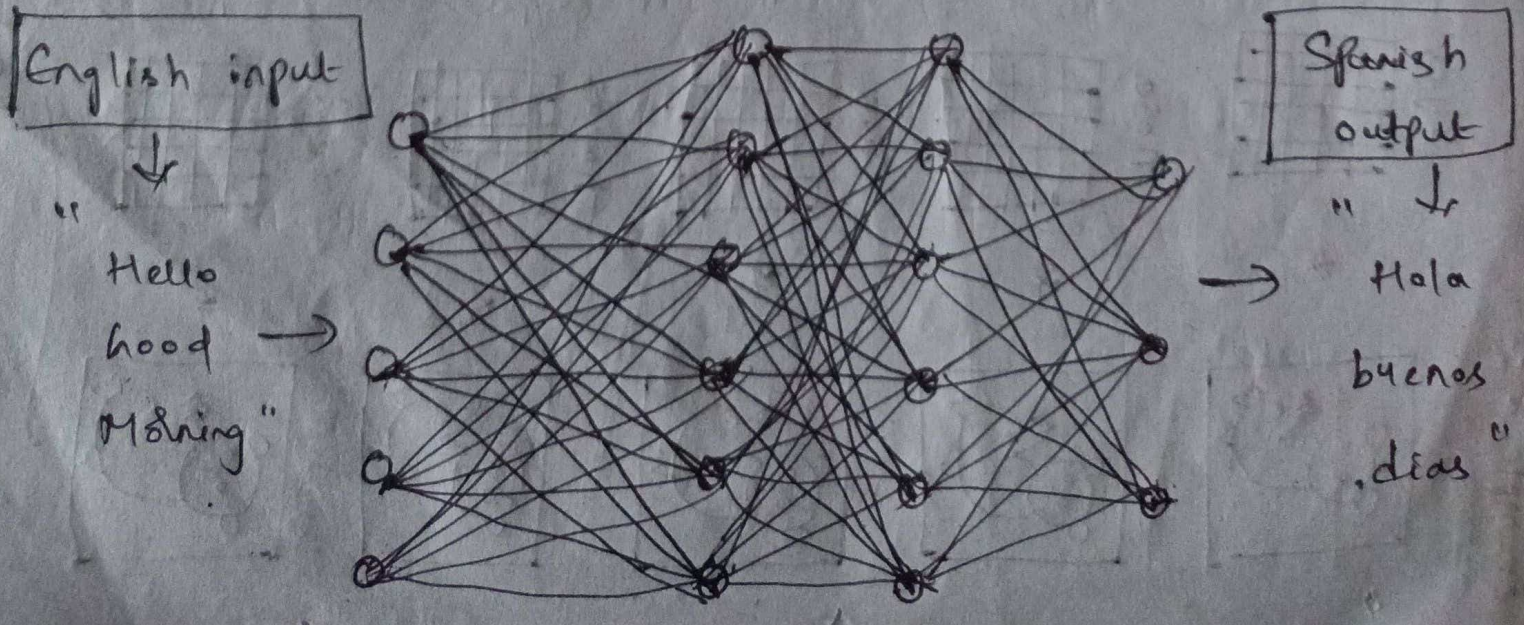

Eyes
nose
Ears
hands
legs
Filters
head
body
Flattening
Deeply Connected full network
is this a Panda ?
Feature Extraction
Feature classification

④ Recurrent Neural Networks (RNN) :-

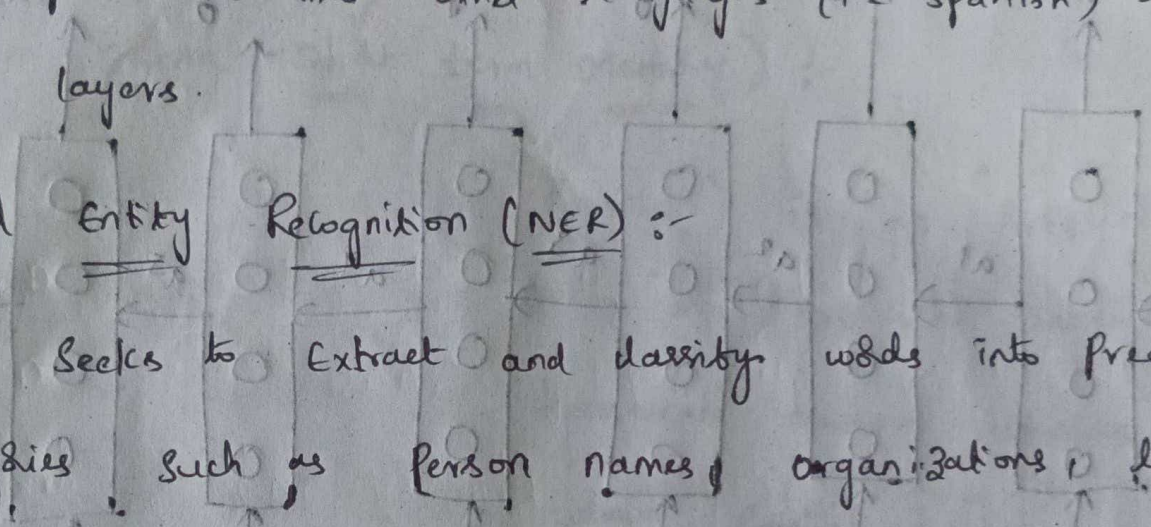
(a) Introduction to RNN :-

⇒ Recurrent Neural Network (RNN) is a type of Neural Network where the output from the previous step are fed as input to the current step. In traditional neural networks, all the inputs and outputs are independent of each other, but in cases like when it is required to predict the next word of a sentence, the previous words are required and hence there is a need to remember the previous words.

⇒ Thus, RNN came into existence, which solved the issue with the help of a Hidden layer. The main and most important feature of RNN is Hidden state, which remembers some information about a sequence



⇒ In the above Example RNN is Particularly used for the Sentence translations and Speech recognition. where we have given an input in English language and we translated the input text into other languages (i.e spanish) by using hidden layers.



⇒ Named Entity Recognition (NER) :-

⇒ NER Seeks to Extract and classify words into Predefined Categories such as Person names, organizations, locations, medical codes, time Expressions, Quantities, monetary values, &c.

⇒ NER Can be used in natural language Processing (NLP) to help answer real-world Problems.

Example 1:-

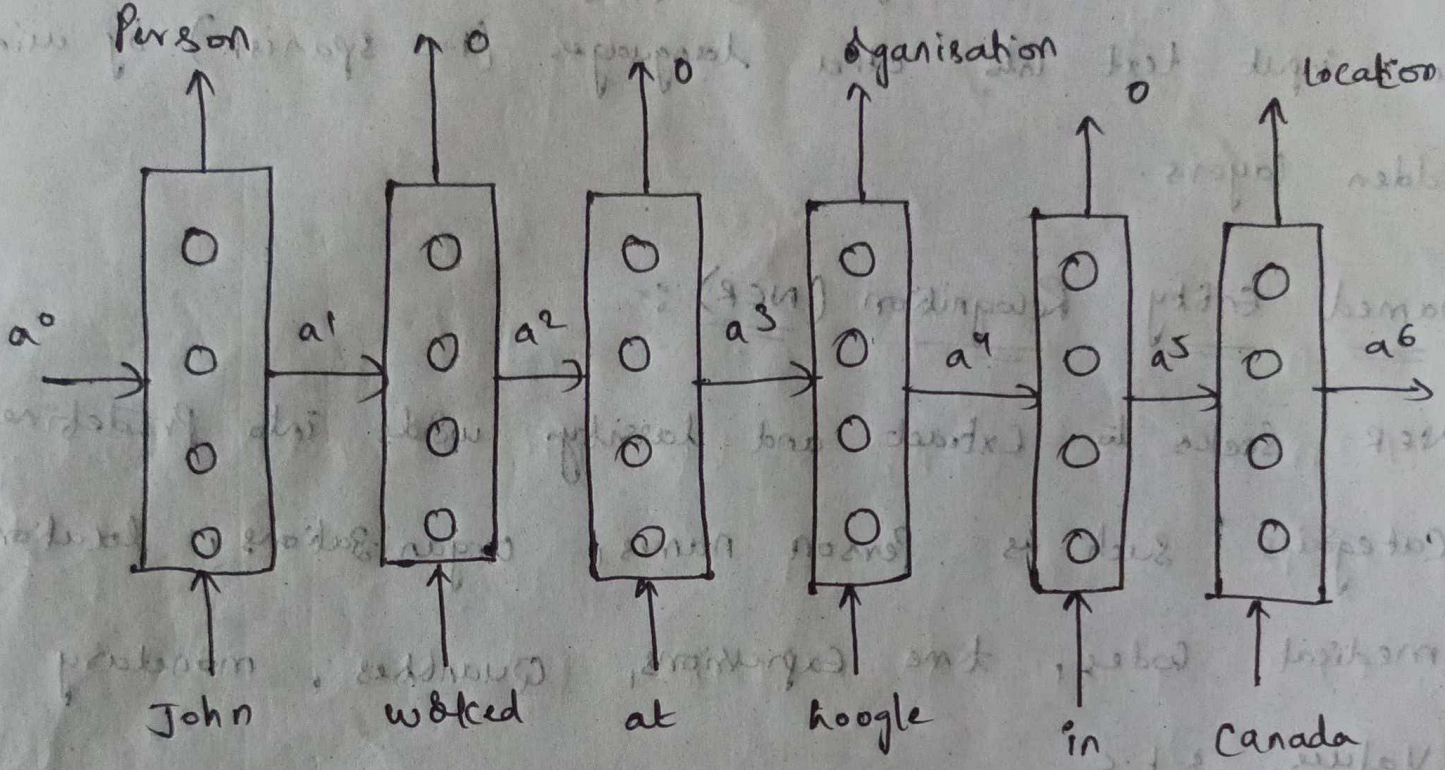
Person — John worked at Google (organisation) in Canada (location)

Example 2:-

Hey, Show me the best places to visit in Mumbai (location) during Jan 2022 (date)

Consider Example 1 :-

John worked at hoogle in Canada



⇒ In the above Example we have to recognize the Entities i.e (attributes Person name, Organisation, location E.t.c) without that Entities the Sentence can't be framed to the next words, only Entities has been recognized remaining we have to be considered as Zero ('O')

⇒ In that Example without the Person we can't frame the words like Organisation and Vice Versa without Organisation we can Unable to frame the word location. i.e the words are dependent on other words

(4b) RNN Code :-

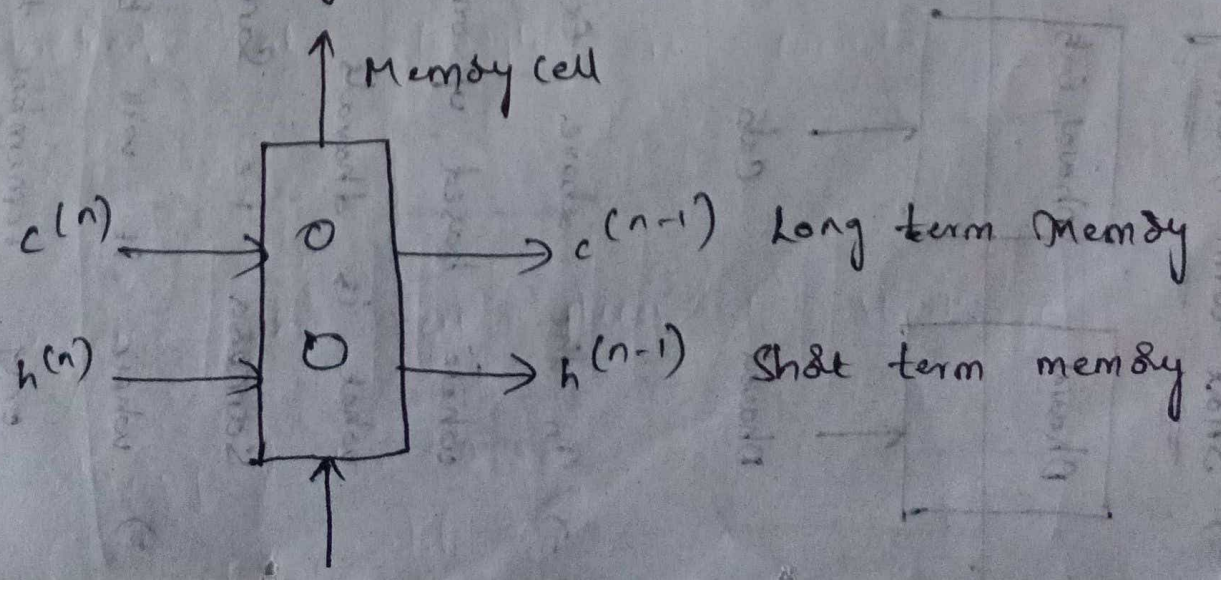
⇒ Two main aspects in Recurrent Neural Network are

1. LSTM 2. GRU

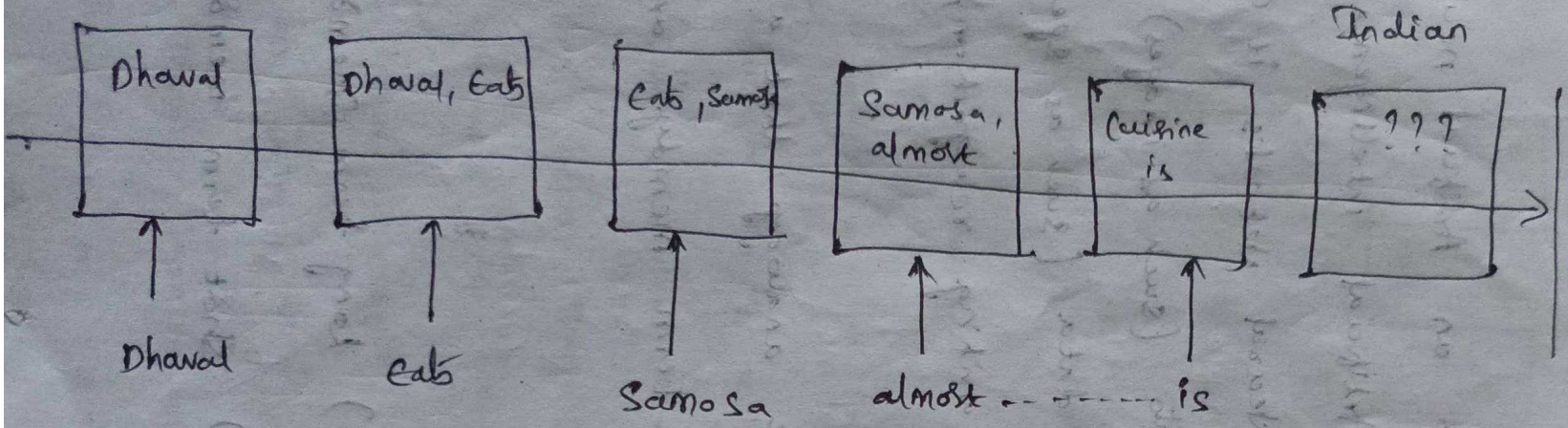
1. LSTM (Long - short term memory) :-

⇒ Long short-term memory (LSTM) is an Artificial neural network used in the fields of Artificial intelligence and deep learning. in Recurrent Neural Networks it can process not only single data points (such as images) but also Entire Sequences of text data (such as speech recognition, Machine translation, handwriting recognition, robot control E.t.c)

⇒ The name of LSTM refers to the analogy that a Standard RNN has both " long-term memory " and " short-term memory ".



⇒ Short term Memory :-



⇒ In the above example each word will be taken as input where based up on the sentence we have to find that what is dhaval's favourite cuisine? it will find based on Samosa i.e Samosa is a Indian Cuisine

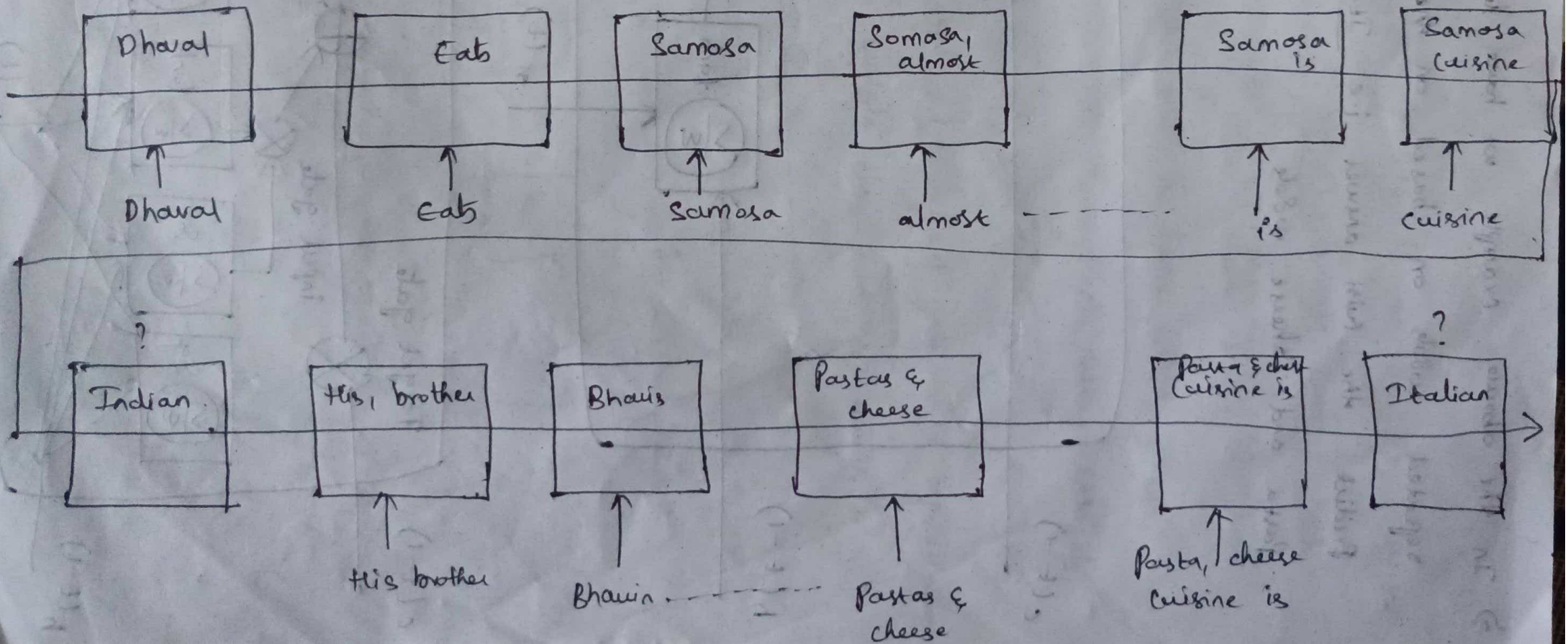
⇒ where it will be based on short-term memory it will only remember the words which is useful for the answer. So, in the above example it will forget all words and remember the only word Samosa and given a result Indian Cuisine

Example :- Consider a sentence and let see how LSTM works

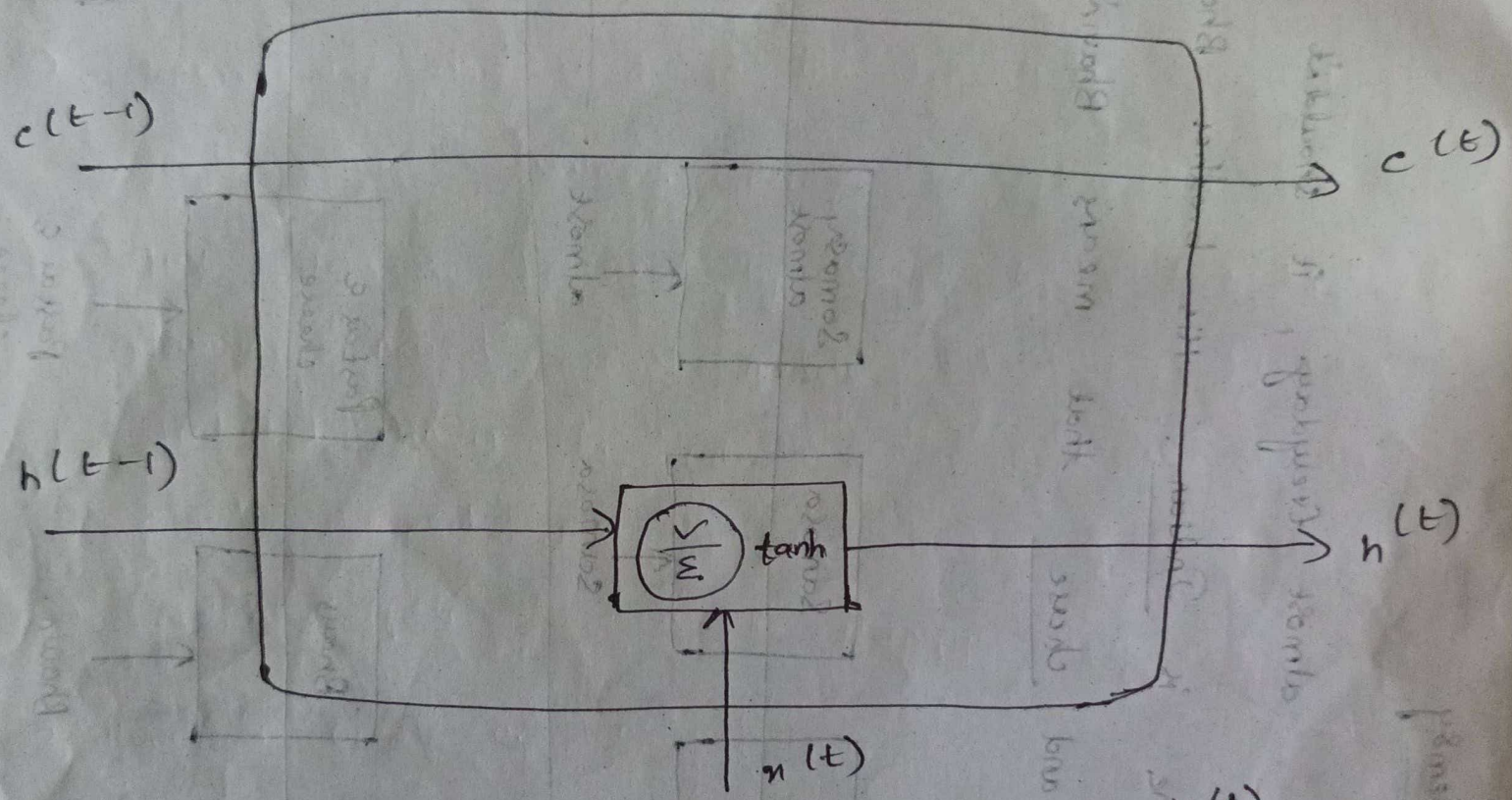
⇒ " Dhaval Eats Samosa almost everyday, it shouldn't be hard to guess that his favourite Cuisine is Indian "?

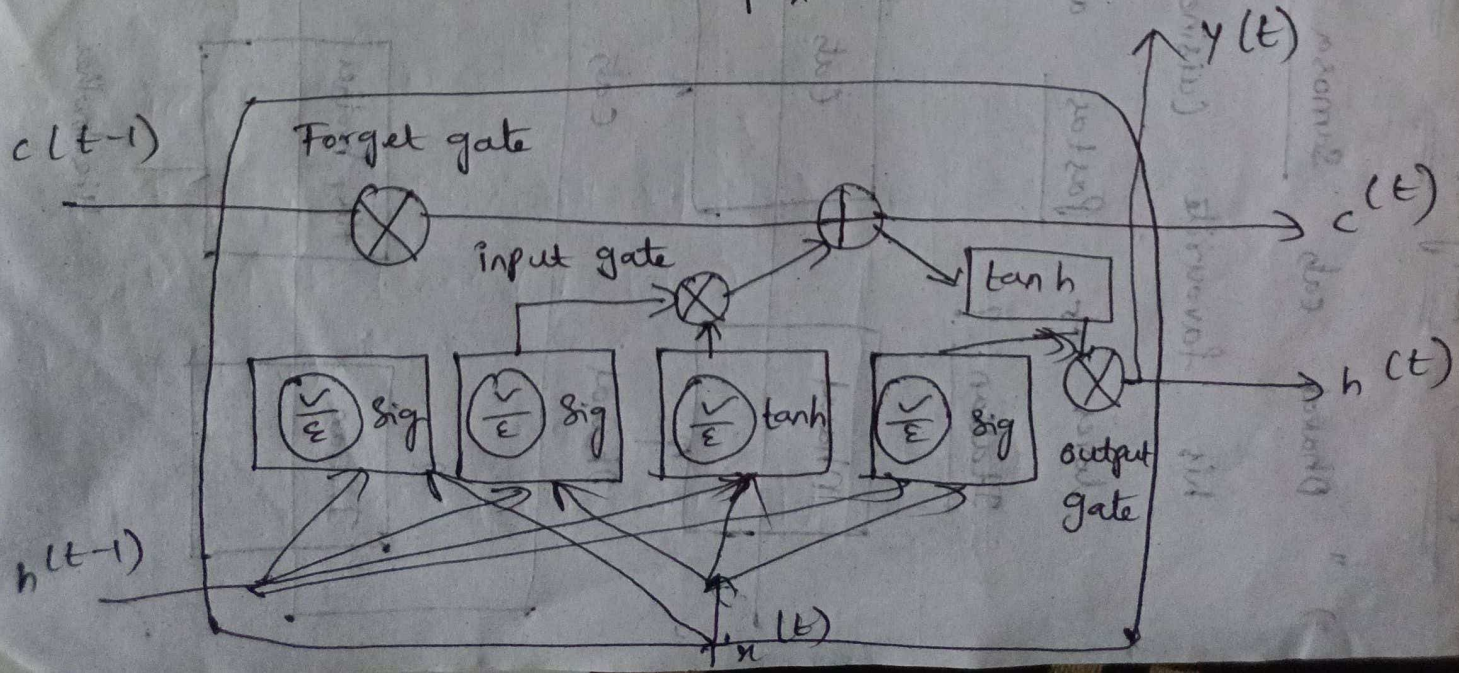
Example 2 :- Long-term Memory

⇒ " Dhaval eats Samosa almost everyday, it shouldn't be hard to guess that his favourite cuisine is Indian. His brother Bhavin however is a lover of Pastas and cheese that means Bhavin's favourite cuisine is Italian ?



⇒ In the above example we have to remember previously repeated words on based on those words we have to predict the wish answer i-e Italian it is based on pasta and cheese words.



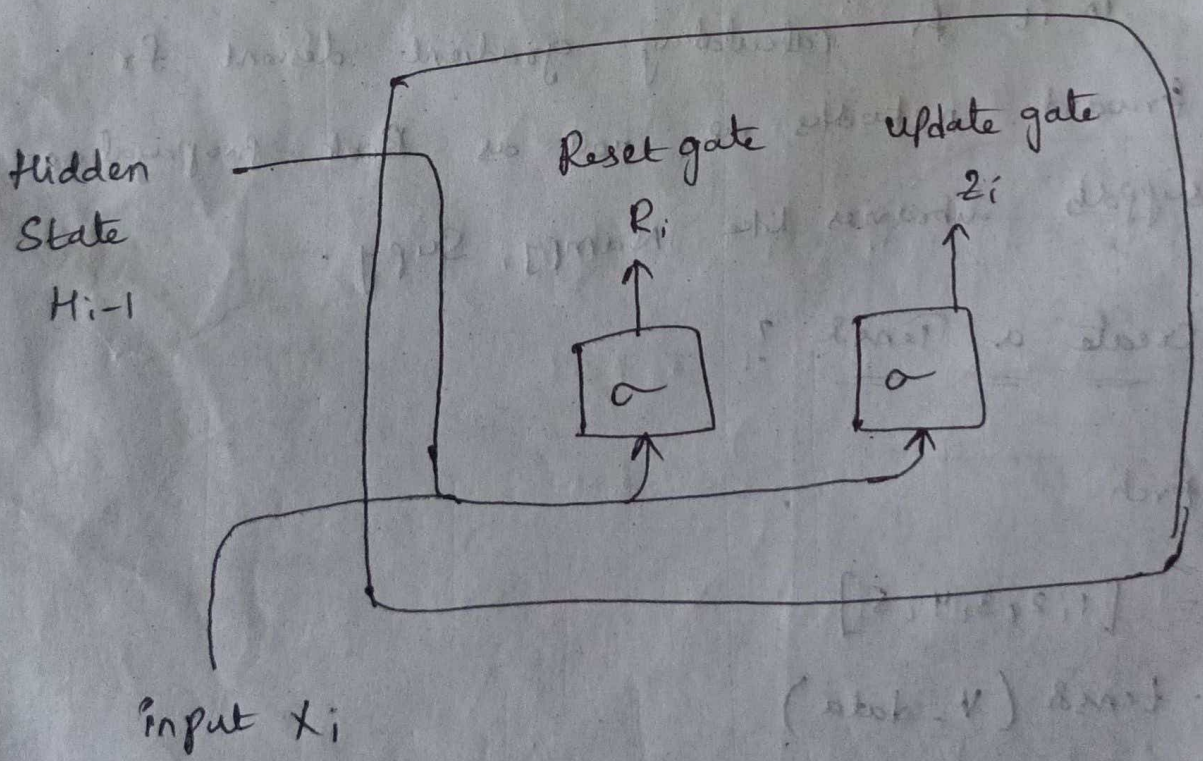


2. GRU (Gated Recurrent Units) :-

⇒ The Gated Recurrent Unit (GRU) is a type of Recurrent Neural Network (RNN) that, in certain cases, has advantages over long short term memory (LSTM).

⇒ GRU uses less memory and is faster than LSTM, however, LSTM is more accurate when using datasets with longer sequences.

⇒ In this we have two gates i.e Reset gate and update gate to reset the data and update the weights of the data



σ fc layer with activation function

copy

Concatenate

③ Pytorch Tensors :-

⇒ A Pytorch Tensor is basically the same as a numpy array. but the biggest difference between a numpy array and a Pytorch array Tensor is that a Pytorch Tensor can run on either CPU & GPU

⇒ Tensors are also used in the Tensorflow framework, which google released. Numpy Arrays in Python are basically Jax Tensors Processed by using GPU's (or) TPU's for training neural network models. Pytorch has libraries included in it for calculating gradient descent for feed-forward networks as well as Back-propagation.

⇒ PyTorch supports libraries like Numpy, Scipy.

⇒ How to create a Tensor ?

```
import torch
V_data = [1, 2, 3, 4, 5]
V = torch.tensor(V_data)
Print(V)
```

output :- tensor([1, 2, 3, 4, 5])

Example :-

Scalar	Vector	matrix	Tensor
1	$\begin{bmatrix} 1 \\ 2 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$	$\begin{bmatrix} [1,2] & [3,2] \\ [1,7] & [5,4] \end{bmatrix}$

⇒ 3

A = torch.tensor(3)

0 dimensions

1.0
2.0
3.0

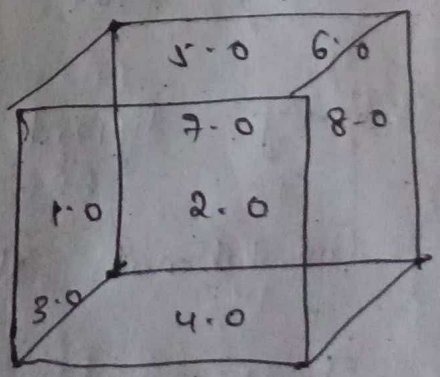
B = torch.tensor([1.0, 2.0, 3.0])

1 Dimension

1.0 2.0
3.0 4.0

C = torch.tensor
([[1.0, 2.0], [3.0, 4.0]])

2 Dimension



D = torch.tensor

([[[[1.0, 2.0], [3.0, 4.0]],
[[5.0, 6.0], [7.0, 8.0]]]])

3 Dimension

3(a) Deep learning with Pytorch :-

⇒ As discussed Previously about Pytorch in this we usually discuss about the phases included in deep learning

⇒ Phases are :-

1. Introduction to PyTorch
2. Build your first Multilayer Perceptron model
3. Training a PyTorch model
4. Using a PyTorch model for reference
5. Loading data from TorchVision
6. Using PyTorch dataLoader
7. Convolutional Neural Network
8. Train an image classifier
9. Train with GPU

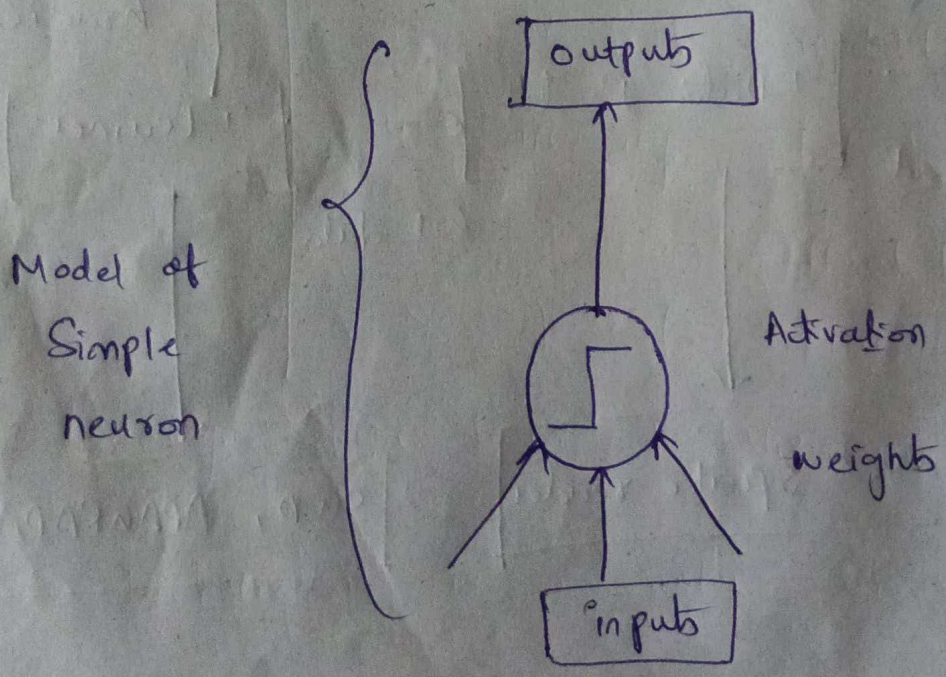
⇒ Introduction to PyTorch :-

⇒ Refer Previous Introduction to PyTorch Series Topic

2) Build your First Multilayer Perceptron (MLP) model :-

⇒ Deep learning is about Building large scale neural networks. The Simplest form of neural network is called multilayer Perceptron model.

⇒ These are Simple computational Units that have weighed input Signals and Produce an output Signal Using an activation function.



⇒ PyTorch allows you to develop and Evaluate deep learning models in very few lines of code.

```
1. import torch.nn as nn
2. model = nn.Sequential(
3.     nn.Linear(8, 12),
```

```
4.    nn.ReLU(),
5.    nn.Linear(12,8),
6.    nn.ReLU(),
7.    nn.Linear(8,1),
8.    nn.Sigmoid()
9.    )
10.   print(model)
```

3) Training a Pytorch model :-

⇒ Building a neural network in Pytorch does not tell how you should train the model for a particular job.

⇒ we can train the model by following below steps.

1. what is the dataset, specifically how the input and the target looks like
2. what is the loss function to evaluate the goodness of fit the model to the data
3. what is the optimization algorithm to train the model, and the parameters to the optimization algorithm such as learning rate and the number of iterations to train.

4) Using a PyTorch Model for reference :-

⇒ A trained neural network model is a model that remembered how the input and target related Then, this model can predict the target given another input.

```
1. model.eval()
2. with torch.no_grad():
3.     y_pred = model(x)
4. accuracy = (y_pred.round() == y).
                float().mean()
5. print(f"Accuracy {accuracy}")
```

5. Loading data from TorchVision :-

⇒ TorchVision is a sister library to PyTorch. In this library, there are functions specialized for image and Computer Vision.

⇒ In this we used to build a deep learning model to classify small images. This is a model that allows your computer to see whats on an image.

```
1. import matplotlib.pyplot as plt
2. import torchvision
3. trainset = torchvision.datasets.CIFAR10(root =
        './data', train = True, download = True)
4. testset = torchvision.datasets.CIFAR10(root =
        './data', train = False, download = True)
5. fig, ax = plt.subplots(4, 6, sharex = True,
                          sharey = True,
                          figsize = (12,8))
6. for i in range(0,24):
       row, col = i//6, i%6
7:     ax[row][col].imshow(trainset.data[i])
8.     plt.show()
```

6) Using PyTorch dataloader :-

⇒ The image from the previous lesson is indeed in the format of numpy array. But for consumption by a PyTorch model, it needs to be in PyTorch

⇒ It is not difficult to convert a numpy array into PyTorch tensor but in the training loop, you still need to divide the dataset in batches. The PyTorch DataLoader can help you make this Process Smoother.

3b) CNN in PyTorch :-

⇒ Convolutional Neural Network :-

⇒ Images are 2D Structures. You can Easily convert them into 1D Vectors by flattening it and build a neural network model to classify them.

⇒ The Standard way for image Processing neural Network is to use Convolutional layers. A neural network that uses Convolutional layers is called a Convolutional neural network. An Example is as follows :-

```
1. import torch.nn as nn
2. model = nn.Sequential(
```

```
3.  nn.Conv2d(3, 32, kernel_size=(3,3),
                Stride = 1, Padding = 1);
4.  nn.ReLU(),
5.  nn.Dropout(0.3),
6.  nn.ConV2d(32, 32, kernel_size = (3,3),
                Stride = 1, Padding = 1),
7.  nn.ReLU(),
8.  nn.MaxPool2d(kernel_size = (2,2)),
9.  nn.Flatten(),
10. nn.Linear(8192, 512)
11. nn.ReLU(),
12. nn.Dropout(0.5),
13. nn.Linear(512, 10)
14. )
15. print(model)
```

Lesson 08: Train an Image Classifier

Together with the DataLoader created for CIFAR-10 dataset, you can train the convolutional neural network in the previous lesson with the following training loop:

```
import torch.nn as nn
import torch.optim as optim

loss_fn = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)

n_epochs = 20
for epoch in range(n_epochs):
    model.train()
    for inputs, labels in trainloader:
        y_pred = model(inputs)
        loss = loss_fn(y_pred, labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    acc = 0
    count = 0
    model.eval()
    with torch.no_grad():
        for inputs, labels in testloader:
            y_pred = model(inputs)
            acc += (torch.argmax(y_pred, 1) == labels).float().sum()
            count += len(labels)
    acc /= count
    print("Epoch %d: model accuracy %.2f%%" % (epoch, acc*100))
```

This will take a while to run, and you should see the model produced can achieve no less than 70% accuracy.

This model is a multiclass classification network. The output is not one, but many scores, one for each class. We consider the higher score the more confident the model thinks the image belongs to a class. The loss function used is therefore **cross-entropy**, the multiclass version of binary cross-entropy.

In the training loop above, you should see quite many elements you learned in the previous lessons. Including switching between training and inference mode in the model, using `torch.no_grad()` context, and calculation of the accuracy.

Lesson 09: Train with GPU

The model training you did in the previous lesson should take a while. If you have a supported GPU, you can speed up the training a lot.

The way to use GPU in PyTorch is to send the model and data to GPU before execution. Then you have an option to send back the result from GPU, or perform the evaluation in GPU directly.

It is not difficult to modify the code from the previous lesson to use GPU. Below is what it should be done:

```
import torch.nn as nn
import torch.optim as optim

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model.to(device)

loss_fn = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)

n_epochs = 20
for epoch in range(n_epochs):
    model.train()
    for inputs, labels in trainloader:
        y_pred = model(inputs.to(device))
        loss = loss_fn(y_pred, labels.to(device))
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    acc = 0
    count = 0
    model.eval()
    with torch.no_grad():
        for inputs, labels in testloader:
            y_pred = model(inputs.to(device))
            acc += (torch.argmax(y_pred, 1) == labels.to(device)).float().sum()
            count += len(labels)
    acc /= count
    print("Epoch %d: model accuracy %.2f%%" % (epoch, acc*100))
```

The changes made are the following: You check if GPU is available and set the `device` accordingly. Then the model is sent to the device. When the input (i.e., a batch of images) is pass on to the model, it should be sent to the corresponding device first. Since the model output will also be there, the loss calculation or the accuracy calculation should also have the target sent to the GPU first.