Lab # 7: Neural Networks

7.1 Objectives

Attempt to implement a simple Neural Network using guided examples to grasp foundational concepts and architecture in neural modelling.

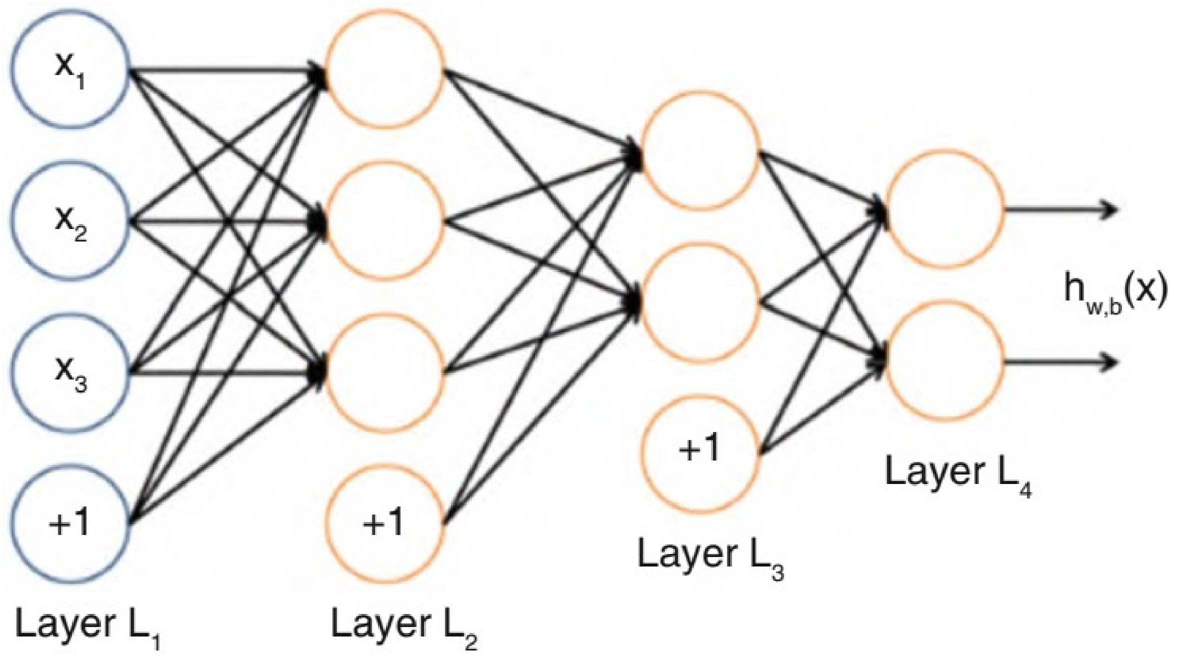
7.2 Pre-Lab

7.2.1 Neural Networks

Neural networks represent deep learning using artificial intelligence. Certain application scenarios are too heavy or out of scope for traditional machine learning algorithms to handle. As they are commonly known, Neural Network pitches in such scenarios and fills the gap.

Neural networks are the representation we make of the brain: neurons interconnected to other neurons, which forms a network. A simple information transits in a lot of them before becoming an actual thing, like “move the hand to pick up thispencil.”

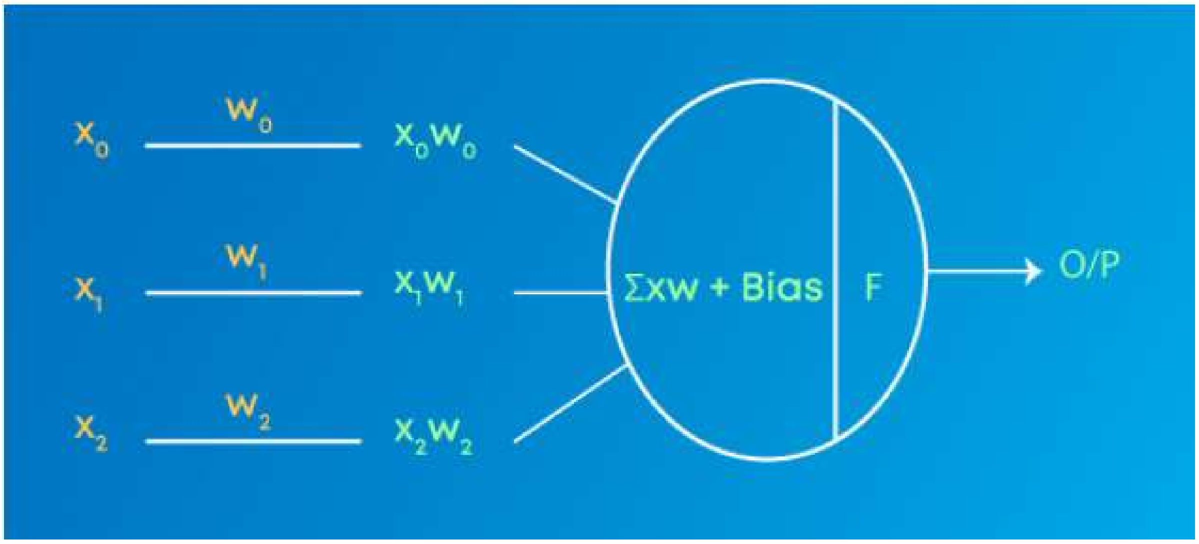
The operation of a complete neural network is straightforward: one enters variables as inputs (for example, an image if the neural network is supposed to tellwhat is on an image), and after some calculations, an output is returned (probabilityof whether an image is a cat).



When an input is given to the neural network, it returns an output. On the first try, it cannot get the right output by its own (except with luck) and that is why, during the learning phase, every input comes with its label, explaining what output the neural network should have guessed. If the choice is the good one, actual parameters are kept, and the next input is given. However, if the obtained output does not match the label, weights are changed. Those are the only variables that can be changed during the learning phase. This process may be imagined as multiple buttons that are turned into different possibilities every time an input is not guessed correctly. To determine which weight is better to modify, a particular process, called “backpropagation” is done.

7.2.2 Components of Neural Networks

39. **Weights** are numeric values that are multiplied by inputs. In backpropagation, they are modified to reduce the loss. In simple words, weights are machine learned values from Neural Networks. They self-adjust depending on the difference between predicted outputs vs training inputs.
40. **Activation Function** is a mathematical formula that helps the neuron to switch ON/OFF.



41. **The input layer** represents dimensions of the input vector.
42. **The hidden layer** represents the intermediary nodes that divide the input space into regions with (soft) boundaries. It takes in a set of weighted input and produces output through an activation function.
43. **Output layer** represents the output of the neural network.

7.2.3 Types of neural networks models are listed below:

The nine types of neural networks are:

44. Perceptron
45. Feed Forward Neural Network
46. Multilayer Perceptron
47. Convolutional Neural Network
48. Radial Basis Functional Neural Network
49. Recurrent Neural Network
50. LSTM – Long Short-Term Memory

51. Sequence to Sequence Models
52. Modular Neural Network

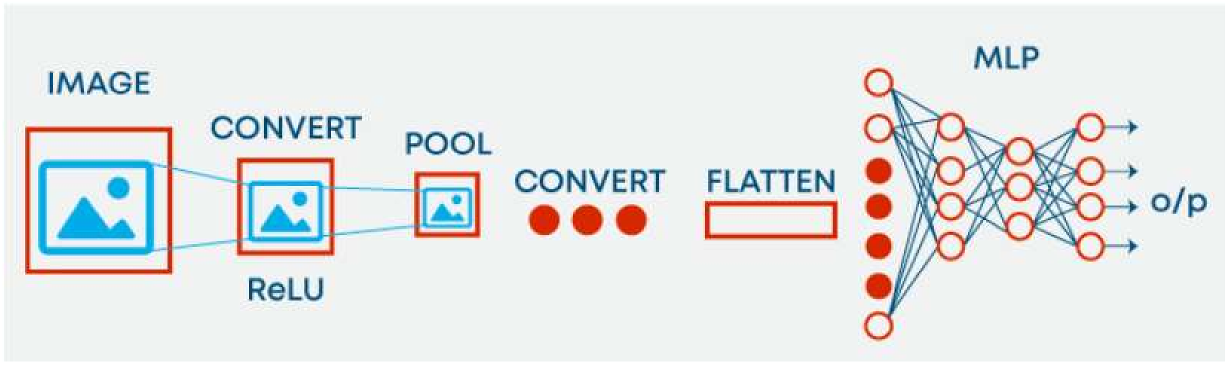
7.2.4 Convolutional Neural Network

Convolution neural network contains a three-dimensional arrangement of neurons instead of the standard two-dimensional array. The first layer is called a convolutional layer. Each neuron in the convolutional layer only processes the information from a small part of the visual field. Input features are taken in batch-wise like a filter. The network understands the images in parts and can compute these operations multiple times to complete the full image processing. Processing involves conversion of the image from RGB or HSI scale to grey scale. Furthering the changes in the pixel value will help to detect the edges and images can be classified into different categories.

These layers perform operations that alter the data with the intent of learning features specific to the data. Three of the most common layers are convolution, activation or ReLU, and pooling.

53. **Convolution** puts the input images through a set of convolutional filters, each of which activates certain features from the images.
54. **Rectified linear unit (ReLU)** allows for faster and more effective training by mapping negative values to zero and maintaining positive values. This is sometimes referred to as *activation*, because only the activated features are carried forward into the next layer.
55. **Pooling** simplifies the output by performing nonlinear downsampling, reducing the number of parameters that the network needs to learn.

These operations are repeated over tens or hundreds of layers, with each layer learning to identify different features.



7.2.4.1 Advantages of Convolution Neural Network:

56. Used for deep learning with few parameters.
57. Less parameters to learn as compared to fully connected layer.

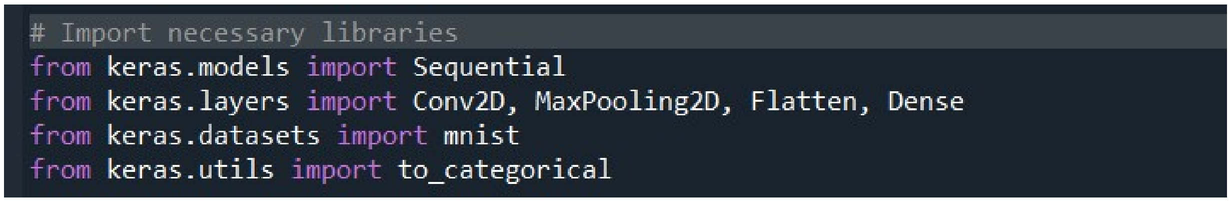
7.2.4.2 Disadvantages of Convolution Neural Network:

58. Comparatively complex to design and maintain.
59. Comparatively slow [depends on the number of hidden layers]

7.3 In-Lab

7.3.1 Implementation of CNN using Python

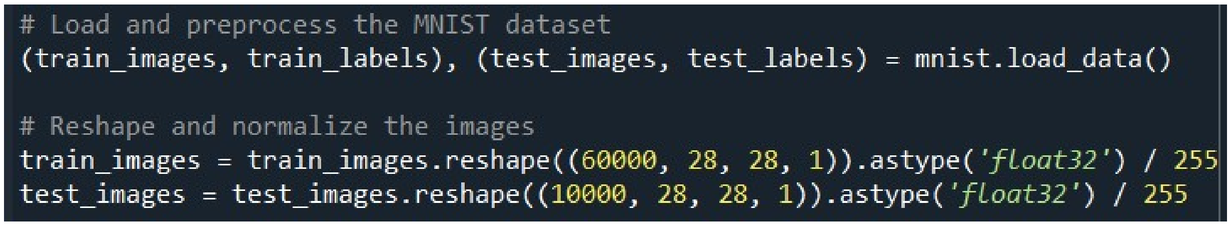
7.3.1.1 Import Necessary Libraries:



```
# Import necessary libraries
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from keras.datasets import mnist
from keras.utils import to_categorical
```

7.3.1.2 Load and Preprocess Data:

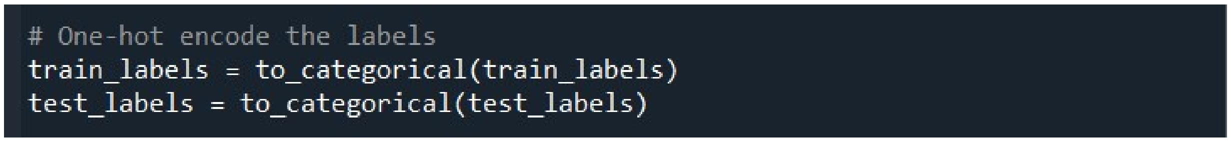
The MNIST dataset is loaded using. mnist.load_data().



```
# Load and preprocess the MNIST dataset
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# Reshape and normalize the images
train_images = train_images.reshape((60000, 28, 28, 1)).astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1)).astype('float32') / 255
```

Images are reshaped to have a single channel (grayscale) and normalized to values between 0 and 1.



```
# One-hot encode the labels
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

7.3.1.3 Build the CNN Model:

The model is created as a sequential stack of layers.

- Convolutional layers (Conv2D) are added with ReLU activation.
- Max pooling layers (MaxPooling2D) are added to down-sample the spatial dimensions.
- A flattening layer (Flatten) is added to convert the 2D feature maps to a vector.
- Dense (fully connected) layers are added with ReLU activation.
- The output layer uses softmax activation for multi-class classification.

```
# Build the CNN model
model = Sequential()

# Step 1: Convolutional Layer with ReLU activation
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))

# Step 2: Max Pooling Layer
model.add(MaxPooling2D((2, 2)))

# Step 3: Convolutional Layer with ReLU activation
model.add(Conv2D(64, (3, 3), activation='relu'))

# Step 4: Max Pooling Layer
model.add(MaxPooling2D((2, 2)))

# Step 5: Flatten Layer
model.add(Flatten())

# Step 6: Dense (Fully Connected) Layer with ReLU activation
model.add(Dense(64, activation='relu'))

# Step 7: Output Layer with Softmax activation (for multi-class classification)
model.add(Dense(10, activation='softmax'))
```

7.3.1.4 Compile the Model:

The model is compiled with the Adam optimizer, categorical cross-entropy loss (appropriate for multi-class classification), and accuracy as the metric to monitor during training.

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

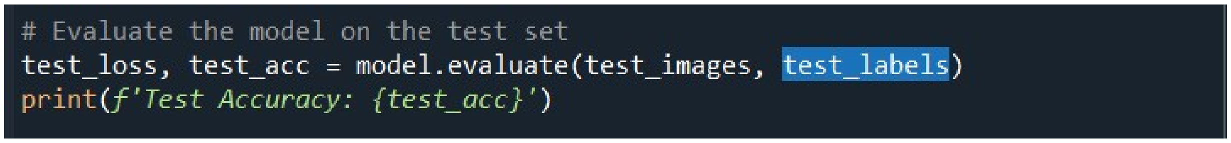
7.3.1.5 Train the Model:

The model is trained using the training data. Adjust the number of epochs and batch size based on your needs.

```
# Train the model
model.fit(train_images, train_labels, epochs=5, batch_size=64, validation_data=(test_images,
                                                                                 test_labels))
```

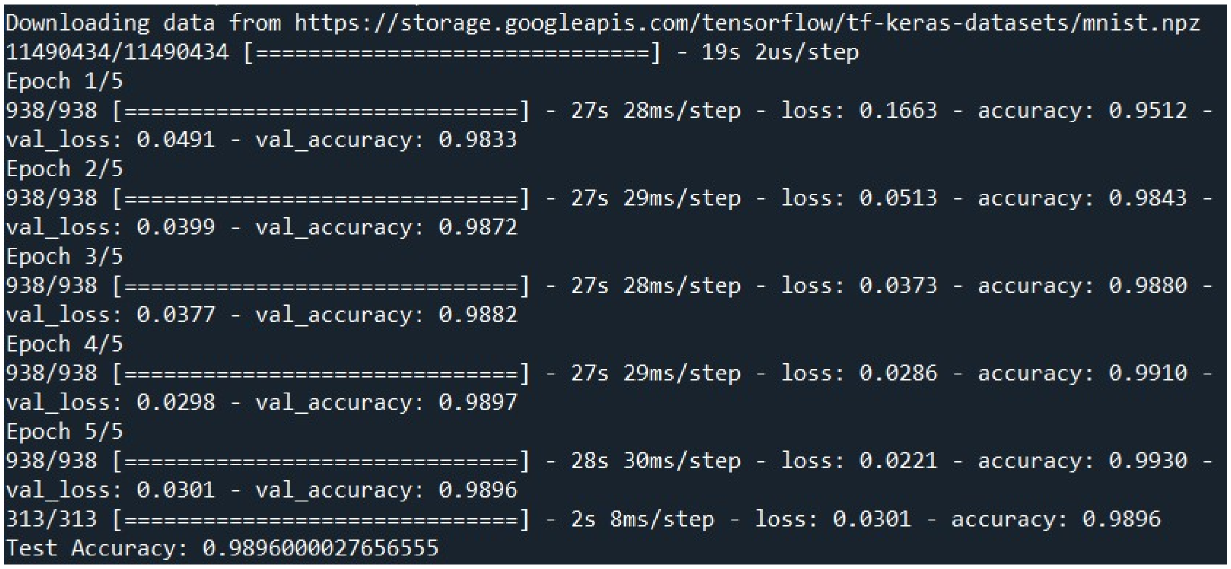
7.3.1.6 Evaluate the Model:

The trained model is evaluated on the test set, and the accuracy is printed.



```
# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f'Test Accuracy: {test_acc}')
```

Results



```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 19s 2us/step
Epoch 1/5
938/938 [==============================] - 27s 28ms/step - loss: 0.1663 - accuracy: 0.9512 -
val_loss: 0.0491 - val_accuracy: 0.9833
Epoch 2/5
938/938 [==============================] - 27s 29ms/step - loss: 0.0513 - accuracy: 0.9843 -
val_loss: 0.0399 - val_accuracy: 0.9872
Epoch 3/5
938/938 [==============================] - 27s 28ms/step - loss: 0.0373 - accuracy: 0.9880 -
val_loss: 0.0377 - val_accuracy: 0.9882
Epoch 4/5
938/938 [==============================] - 27s 29ms/step - loss: 0.0286 - accuracy: 0.9910 -
val_loss: 0.0298 - val_accuracy: 0.9897
Epoch 5/5
938/938 [==============================] - 28s 30ms/step - loss: 0.0221 - accuracy: 0.9930 -
val_loss: 0.0301 - val_accuracy: 0.9896
313/313 [==============================] - 2s 8ms/step - loss: 0.0301 - accuracy: 0.9896
Test Accuracy: 0.9896000027656555
```