| **Ex No: 2**  **Date: Aug 14** | **Implementing a Multi-Class Classification Model using Deep neural network** |
| --- | --- |

# Objective:

The objective of this lab is to develop a deep neural network (DNN) model to accurately classify images of flowers into five distinct categories: daisy, dandelion, rose, sunflower, and tulip. We have to train and evaluate the model on a provided dataset, using multiple layers of the neural network to learn meaningful patterns in the image data and to achieve optimal performance in recognizing different types of flowers.

# Description:

Multi-class classification is a type of classification task where a model predicts a label for an input from three or more distinct classes. Unlike binary classification, which deals with two classes, multi-class classification involves predicting one class out of a set of multiple possible categories.

In multi-class classification, an input sample belongs to one of **N** possible classes. For example, in this lab, the goal is to classify flowers into one of five classes: daisy, dandelion, rose, sunflower, or tulip.

Using a deep neural network (DNN) for flower classification involves employing softmax activation in the output layer, utilizing categorical cross-entropy as the loss function, and measuring performance with metrics such as accuracy or F1-score across the five classes: daisy, dandelion, rose, sunflower, and tulip.

# Model:

The deep learning model used for this flower classification task is a deep neural network (DNN). This model is designed to learn complex patterns in image data and classify each image into one of five categories: daisy, dandelion, rose, sunflower, and tulip.

## Data Preparation

* **Data normalization**: Images are usually represented as pixel values between 0 and 255. Normalizing the pixel values to a range between 0 and 1 helps the model train more efficiently.
* **One-hot encoding**: Since this is a multi-class problem, labels (e.g., daisy, dandelion, etc.) need to be one-hot encoded. For five classes, the label vector for each example would be of length 5 (e.g., daisy could be [1, 0, 0, 0, 0], dandelion could be [0, 1, 0, 0, 0], and so on).

## Deep Neural Network Architecture

The DNN model for image classification includes several layers:

* **Input Layer**: Receives the image data. For instance, here the images are 128x128 RGB, the input layer will have 49152 units (128x128x3).
* **Hidden Layers**: These consist of fully connected layers (dense layers). Each neuron in a layer is connected to every neuron in the next layer. It’s common to use activation functions like **ReLU** (Rectified Linear Unit) in hidden layers to introduce non-linearity, enabling the network to learn complex patterns.
* **Output Layer**: For multi-class classification, the output layer typically has one neuron per class,therefore this model has 5 neurons and the activation function used is **softmax**, which outputs probabilities for each class.

## **Forward Propagation**:

* Data is fed into the input layer.
* It is passed through hidden layers where each neuron computes a weighted sum of inputs, applies a bias term, and uses an activation function.
* The output layer produces a probability distribution over the classes using the softmax function.

## **Backward Propagation**:

* Computes the gradient of the loss function with respect to each weight in the network.
* Gradients are propagated backward through the network, from the output layer to the input layer, using the chain rule.
* The optimizer uses these gradients to update the weights and biases.

## Hyperparameter Choices

* **Learning Rate**: This determines how much to adjust the weights during training. A common starting point is 0.001.
* **Batch Size**: The number of samples processed before the model is updated.
* **Epochs**: The number of times the model iterates over the entire training dataset.

## Compiling the Model

After defining the architecture, you need to compile the model. This involves:

* **Loss function**: For multi-class classification, the typical loss function is **categorical cross entropy**. It measures the dissimilarity between the true label distribution and the predicted probability distribution.
* **Metrics**: Accuracy is a common metric for classification problems, but you can also track others like precision, recall, or F1-score.

## 5. Training the Model

The model function represents the core process of training a deep neural network. It begins by initializing the parameters of the network, such as weights and biases, based on the structure (number of layers and neurons) provided in layers\_dims. The training happens over multiple iterations, during which the model performs the following steps in each iteration:

* **Forward Propagation**: The input data (X) is passed through the network to produce predictions (AL). This step computes the activations of neurons in each layer based on the current parameters.
* **Cost Calculation**: The model calculates how far off its predictions are from the actual labels (Y). This is done using a loss function, typically cross-entropy for classification tasks.
* **Backward Propagation**: This step involves calculating the gradients of the cost function with respect to each parameter (weights and biases). The gradients indicate how much each parameter should change to reduce the error.
* **Parameter Update**: Using the computed gradients, the model updates its parameters (weights and biases) to improve its predictions. This is typically done using an optimization algorithm like gradient descent, controlled by the learning rate.

The process repeats for the specified number of iterations (num\_iterations), and the cost is recorded periodically to monitor how the model is learning. At the end of training, the function returns the learned parameters, which can be used to make predictions on new data.

# Evaluating the Model

After training, evaluate the model on the test set to gauge its generalization ability. This involves computing accuracy and other metrics like confusion matrix, precision, recall, and F1-score.

Example:

**Confusion Matrix**: This is particularly useful in multi-class classification as it provides insight into how well the model is performing across different classes.

Example:

# Results

## Performance on the Training and Test Sets

## **Training Accuracy:** 99.2%

* **Test Accuracy:** 44.0%

## Evaluation Metrics

* **Accuracy:** 77.60000000000001%
* **Precision**: 44.0%
* **Recall:**  44.0%