| Experiment No. 6 |
| --- |
| Design and implement a CNN model for Object Detection using CIFAR-10 dataset |
| Date of Performance: |
| Date of Submission: |

**Aim:** Design and implement a CNN model for Object Detection using CIFAR-10 dataset.

**Objective:** Ability to design convolution neural network to solve the given problem

**Theory:**

A Convolutional Neural Network (CNN) is a type of Deep Learning neural network architecture commonly used in Computer Vision. Computer vision is a field of Artificial Intelligence that enables a computer to understand and interpret the image or visual data.
When it comes to Machine Learning, Artificial Neural Networks perform really well. Neural Networks are used in various datasets like images, audio, and text. Different types of Neural Networks are used for different purposes, for example for predicting the sequence of words we use Recurrent Neural Networks more precisely an LSTM, similarly for image classification we use Convolution Neural networks. In this blog, we are going to build a basic building block for CNN.

In a regular Neural Network there are three types of layers:

Input Layers: It's the layer in which we give input to our model. The number of neurons in this layer is equal to the total number of features in our data (number of pixels in the case of an image).
Hidden Layer: The input from the Input layer is then feed into the hidden layer. There can be many hidden layers depending upon our model and data size. Each hidden layer can have different numbers of neurons which are generally greater than the number of features. The output from each layer is computed by matrix multiplication of output of the previous layer with learnable weights of that layer and then by the addition of learnable biases followed by activation function which makes the network nonlinear.
Output Layer: The output from the hidden layer is then fed into a logistic function like sigmoid or softmax which converts the output of each class into the probability score of each class.
The data is fed into the model and output from each layer is obtained from the above step is called feedforward, we then calculate the error using an error function, some common error functions are cross-entropy, square loss error, etc. The error function measures how well the

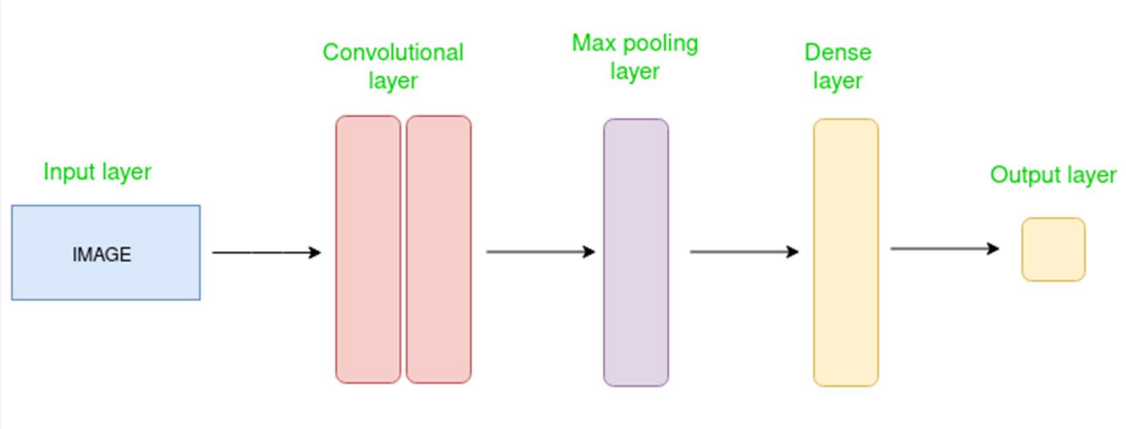
network is performing. After that, we backpropagate into the model by calculating the derivatives. This step is called Backpropagation which basically is used to minimize the loss.

**Convolution neural network:**

Convolutional Neural Network (CNN) is the extended version of artificial neural networks (ANN) which is predominantly used to extract the feature from the grid-like matrix dataset. For example visual datasets like images or videos where data patterns play an extensive role.

**CNN architecture**

Convolutional Neural Network consists of multiple layers like the input layer, Convolutional layer, Pooling layer, and fully connected layers.




The Convolutional layer applies filters to the input image to extract features, the Pooling layer downsamples the image to reduce computation, and the fully connected layer makes the final prediction. The network learns the optimal filters through backpropagation and gradient descent.

**Layers In CNN:**

Input Layers: It's the layer in which we give input to our model. In CNN, Generally, the input will be an image or a sequence of images. This layer holds the raw input of the image with width 32, height 32, and depth 3.

Convolutional Layers: This is the layer, which is used to extract the feature from the input dataset. It applies a set of learnable filters known as the kernels to the input images. The filters/kernels are smaller matrices usually 2×2, 3×3, or 5×5 shape. it slides over the input image data and computes the dot product between kernel weight and the corresponding input image patch. The output of this layer is referred ad feature maps. Suppose we use a total of 12 filters for this layer we'll get an output volume of dimension 32 x 32 x 12.

Activation Layer: By adding an activation function to the output of the preceding layer, activation layers add nonlinearity to the network. it will apply an element-wise activation function to the output of the convolution layer. Some common activation functions are RELU: max(0, x), Tanh, Leaky RELU, etc. The volume remains unchanged hence output volume will have dimensions 32 x 32 x 12.

Pooling layer: This layer is periodically inserted in the covnets and its main function is to reduce the size of volume which makes the computation fast reduces memory and also prevents overfitting. Two common types of pooling layers are max pooling and average pooling. If we use a max pool with 2 x 2 filters and stride 2, the resultant volume will be of dimension 16x16x12.

Flattening: The resulting feature maps are flattened into a one-dimensional vector after the convolution and pooling layers so they can be passed into a completely linked layer for categorization or regression.

Fully Connected Layers: It takes the input from the previous layer and computes the final classification or regression task.

Output Layer: The output from the fully connected layers is then fed into a logistic function for classification tasks like sigmoid or softmax which converts the output of each class into the probability score of each class.

**Program:**

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import keras
import tensorflow as tf

from tensorflow import keras
from keras.models import Sequential
from tensorflow.keras.layers import Input, Conv2D, Dense, Flatten, Dropout
from tensorflow.keras.layers import GlobalMaxPooling2D, MaxPooling2D
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.models import Model
from tensorflow.keras import regularizers, optimizers
from tensorflow.keras.utils import to_categorical
from sklearn.metrics import accuracy_score

import warnings
warnings.filterwarnings('ignore')

print("Tensorflow version:",tf.__version__)
print("Keras version:",keras.__version__)
from tensorflow.keras.datasets import cifar10
(X_train, Y_train), (X_test, Y_test) = cifar10.load_data()
X_train = X_train/255
X_test = X_test/255

Y_train_en = to_categorical(Y_train,10)
Y_test_en = to_categorical(Y_test,10)



model = Sequential()
model.add(Conv2D(64, (4, 4), input_shape=(32, 32, 3), activation='relu'))
model.add(Conv2D(64, (4, 4), activation='relu'))
```

```
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.4))
model.add(Conv2D(128, (4, 4), activation='relu'))
model.add(Conv2D(128, (4, 4), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.4))
model.add(Flatten())
model.add(Dense(1024, activation='relu'))
model.add(Dense(1024, activation='relu'))
num_classes = 10
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

model.summary()
history = model.fit(X_train, Y_train_en, epochs = 20, verbose=1,validation_data=(X_test,Y_test_en))
```

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d_6 (Conv2D) | (None, 29, 29, 64) | 3,136 |
| conv2d_7 (Conv2D) | (None, 26, 26, 64) | 65,600 |
| max_pooling2d_4 (MaxPooling2D) | (None, 13, 13, 64) | 0 |
| dropout_2 (Dropout) | (None, 13, 13, 64) | 0 |
| conv2d_8 (Conv2D) | (None, 10, 10, 128) | 131,200 |
| conv2d_9 (Conv2D) | (None, 7, 7, 128) | 262,272 |
| max_pooling2d_5 (MaxPooling2D) | (None, 3, 3, 128) | 0 |
| dropout_3 (Dropout) | (None, 3, 3, 128) | 0 |
| flatten_2 (Flatten) | (None, 1152) | 0 |
| dense_5 (Dense) | (None, 1024) | 1,180,672 |
| dense_6 (Dense) | (None, 1024) | 1,049,600 |
| dense_7 (Dense) | (None, 10) | 10,250 |

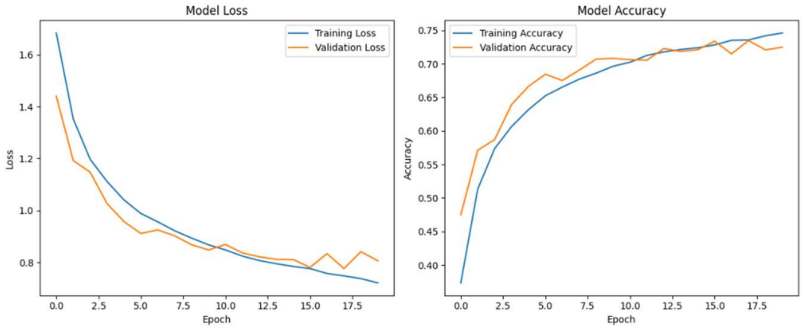
```
Epoch 20/20
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 20s 7ms/step - accuracy: 0.7511 - loss: 0.7073 - val_accuracy: 0.7247 - val_loss: 0.8063
```

```
history_dict = history.history
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history_dict['loss'])
```

```
plt.plot(history_dict['val_loss'])
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Training Loss', 'Validation Loss'])
plt.subplot(1, 2, 2)
plt.plot(history_dict['accuracy'])
plt.plot(history_dict['val_accuracy'])
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Training Accuracy', 'Validation Accuracy'])

plt.tight_layout()
plt.show()
```




**Conclusion:**

Comment on the architecture and of CNN and the results.

Convolutional Neural Networks (CNNs) are deep learning architectures designed for processing images, where they excel at automatically learning hierarchical features through layers of convolution, activation, and pooling. By applying filters to detect local patterns and reducing dimensionality with pooling, CNNs efficiently capture important features, leading to high accuracy in tasks like image classification and object detection. Their ability to generalize well from large datasets has made them a cornerstone in computer vision, though they require significant computational resources and can overfit with limited data.