SAP ID:

SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (DATA SCIENCE)

COURSE CODE: DJ19DSC501 **DATE:6/11/23**

COURSE NAME: Machine Learning - II **CLASS: AY 2022-23**

LAB EXPERIMENT NO.5

Name: Bhuvi Ghosh **SAPID:** 60009210191 **Batch:** D22

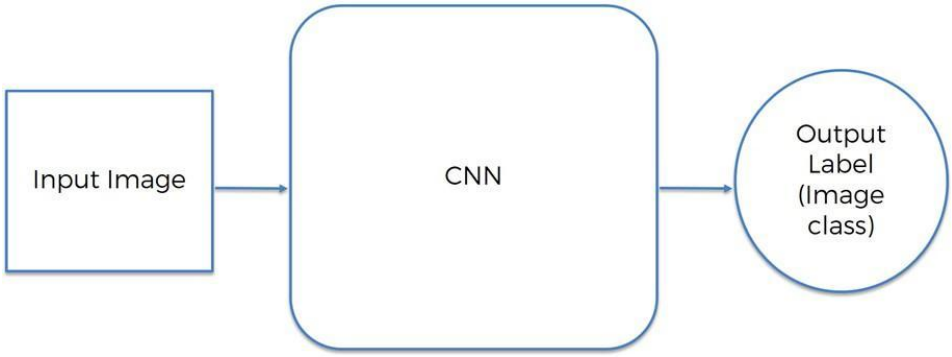
AIM :

Building CNN models for image categorization.

THEORY:

A **Convolutional Neural Network (ConvNet/CNN)** is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics. The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area.

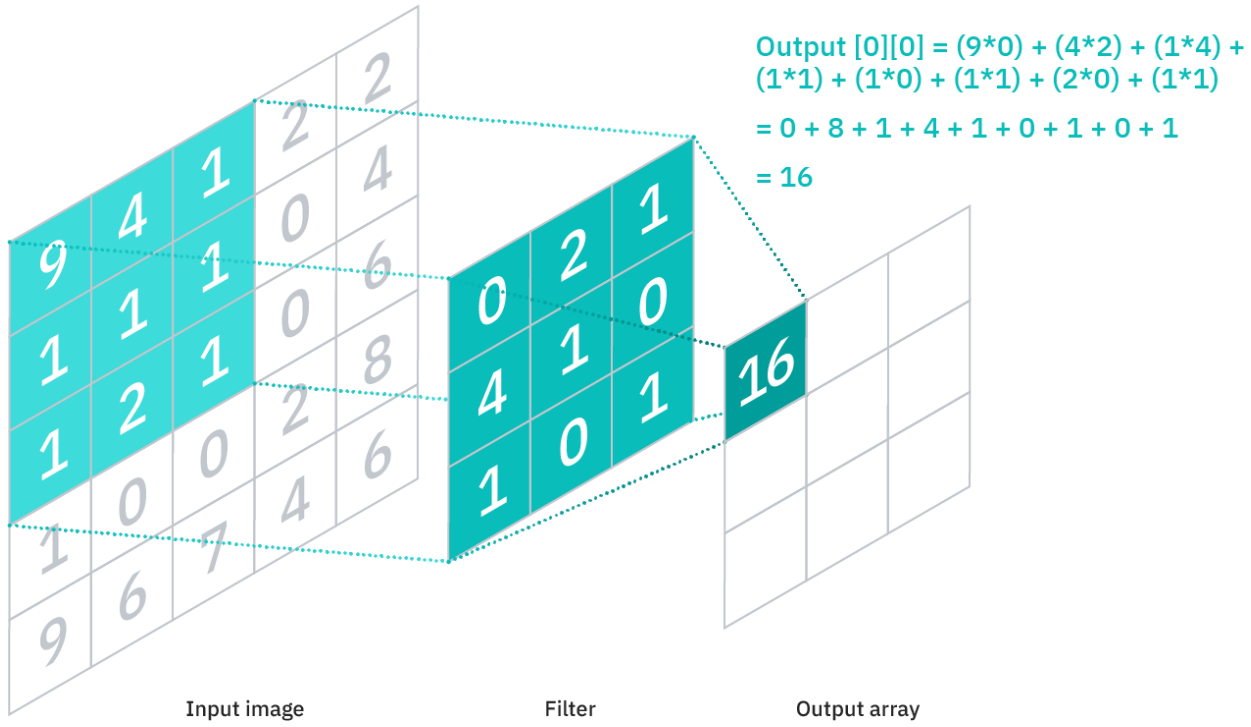
CNN in image categorization base view:



Major Steps in building a CNN:

1) Convolution: The convolutional layer is the core building block of a CNN, and it is where the majority of computation occurs. It requires a few components, which are input data, a filter, and a feature map. Let's assume that the input will be a color image, which is made up of a matrix of

pixels in 3D. This means that the input will have three dimensions—a height, width, and depth—which correspond to RGB in an image. We also have a feature detector, also known as a kernel or a filter, which will move across the receptive fields of the image, checking if the feature is present. This process is known as a convolution. The feature detector is a two-dimensional (2-D) array of weights, which represents part of the image. While they can vary in size, the filter size is typically a 3x3 matrix; this also determines the size of the receptive field. The filter is then applied to an area of the image, and a dot product is calculated between the input pixels and the filter. This dot product is then fed into an output array. Afterwards, the filter shifts by a stride, repeating the process until the kernel has swept across the entire image. The final output from the series of dot products from the input and the filter is known as a feature map, activation map, or a convolved feature.



2) ReLU: After each convolution operation, a CNN applies a Rectified Linear Unit (ReLU) transformation to the feature map, introducing nonlinearity to the model.

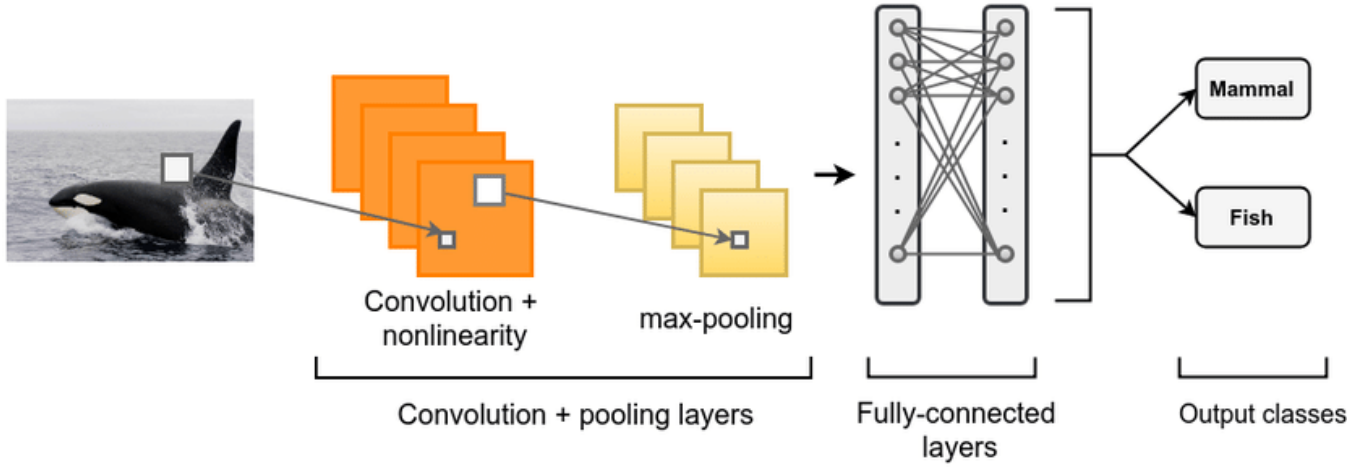
Another convolution layer can follow the initial convolution layer. When this happens, the structure of the CNN can become hierarchical as the later layers can see the pixels within the receptive fields of prior layers.

3) Pooling: Pooling layers, also known as downsampling, 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. Instead, the kernel applies an aggregation function to the values within the receptive field, populating the output array. There are two main types of pooling:

- **Max pooling:** As the filter moves across the input, it selects the pixel with the maximum value to send to the output array. As an aside, this approach tends to be used more often compared to average pooling.
- **Average pooling:** As the filter moves across the input, it calculates the average value within the receptive field to send to the output array.

While a lot of information is lost in the pooling layer, it also has a number of benefits to the CNN. They help to reduce complexity, improve efficiency, and limit risk of overfitting.

4) 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. This layer performs the task of classification based on the features extracted through the previous layers and their different filters. While convolutional and pooling layers tend to use ReLu functions, FC layers usually leverage a softmax activation function to classify inputs appropriately, producing a probability from 0 to 1. Output layer can have a single neuron if problem is a binary classification problem or it can have more than 1 output neurons if problem is a multi-class classification problem.



SAP ID:

SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)

Tasks to be performed:

1. **Choose appropriate Image Categorization dataset**
   (Image Categorization dataset from Kaggle, UCI Machine Learning Repository, Data.Gov or any other online resource/website.)
   OR
   INPUT DATA: https://www.kaggle.com/datasets/paultimothymooney/blood-cells
2. **Build an multiclass image categorization CNN network which correctly classifies different categories of images in the dataset.**
3. **Split original dataset to train and test set**
4. **Generate the accuracy of the built model.**
5. **Perform hyperparameter tuning to increase the accuracy of the CNN.**

Code:

```
[1] !mkdir -p ~/.kaggle
    !cp kaggle.json ~/.kaggle/

[2] !kaggle datasets download -d salader/dogs-vs-cats

    Warning: Your Kaggle API key is readable by other users on this system! To fix this, you can run 'chmod 600 /root/.kaggle/kaggle.json'
    Downloading dogs-vs-cats.zip to /content
     99% 1.06G/1.06G [00:06<00:00, 74.1MB/s]
    100% 1.06G/1.06G [00:06<00:00, 174MB/s]

[3] import zipfile
    zip_ref = zipfile.ZipFile('/content/dogs-vs-cats.zip', 'r')
    zip_ref.extractall('/content')
    zip_ref.close()

[4] import tensorflow as tf
    from tensorflow import keras
    from keras import Sequential
    from keras.layers import Dense,Conv2D,MaxPooling2D,Flatten,BatchNormalization,Dropout
```

```
[5] # generators
    train_ds = keras.utils.image_dataset_from_directory(
        directory = '/content/train',
        labels='inferred',
        label_mode = 'int',
        batch_size=32,
        image_size=(256,256)
    )

    validation_ds = keras.utils.image_dataset_from_directory(
        directory = '/content/test',
        labels='inferred',
        label_mode = 'int',
        batch_size=32,
        image_size=(256,256)
    )

    Found 20000 files belonging to 2 classes.
    Found 5000 files belonging to 2 classes.

[6] # Normalize
    def process(image,label):
        image = tf.cast(image/255. ,tf.float32)
        return image,label

    train_ds = train_ds.map(process)
    validation_ds = validation_ds.map(process)
```

SAP ID:


```
# create CNN model

model = Sequential()

model.add(Conv2D(32,kernel_size=(3,3),padding='valid',activation='relu',input_shape=(256,256,3)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2),strides=2,padding='valid'))

model.add(Conv2D(64,kernel_size=(3,3),padding='valid',activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2),strides=2,padding='valid'))

model.add(Conv2D(128,kernel_size=(3,3),padding='valid',activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2),strides=2,padding='valid'))

model.add(Flatten())

model.add(Dense(128,activation='relu'))
model.add(Dropout(0.1))
model.add(Dense(64,activation='relu'))
model.add(Dropout(0.1))
model.add(Dense(1,activation='sigmoid'))
```

```
[8] model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 254, 254, 32)      896

 batch_normalization (Batch  (None, 254, 254, 32)      128
 Normalization)

 max_pooling2d (MaxPooling2  (None, 127, 127, 32)      0
 D)

 conv2d_1 (Conv2D)           (None, 125, 125, 64)      18496

 batch_normalization_1 (Bat  (None, 125, 125, 64)      256
 chNormalization)

 max_pooling2d_1 (MaxPoolin  (None, 62, 62, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 60, 60, 128)       73856

 batch_normalization_2 (Bat  (None, 60, 60, 128)       512
 chNormalization)

 max_pooling2d_2 (MaxPoolin  (None, 30, 30, 128)       0
 g2D)

 flatten (Flatten)           (None, 115200)            0

 dense (Dense)               (None, 128)               14745728

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 64)                8256

 dropout_1 (Dropout)         (None, 64)                0

 dense_2 (Dense)             (None, 1)                 65

=================================================================
Total params: 14848193 (56.64 MB)
Trainable params: 14847745 (56.64 MB)
Non-trainable params: 448 (1.75 KB)
_________________________________________________________________
```

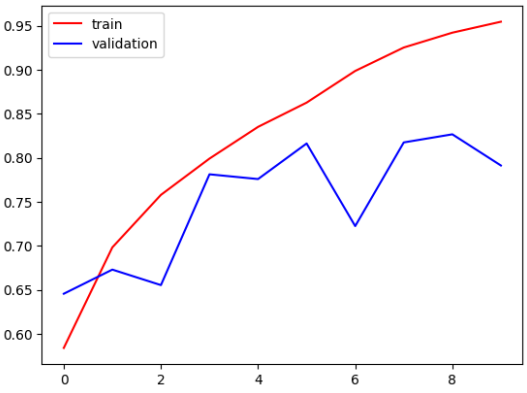
```
[9] model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
```

```
[10] history = model.fit(train_ds,epochs=10,validation_data=validation_ds)
```

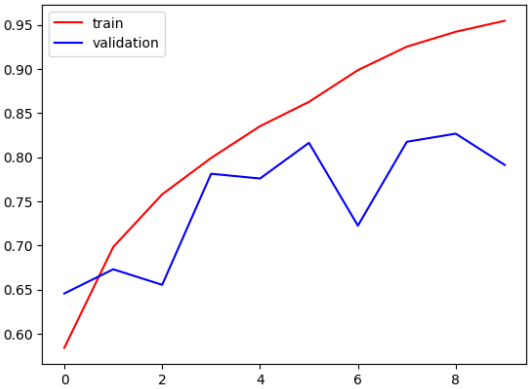
```
Epoch 1/10
625/625 [==============================] - 81s 107ms/step - loss: 1.4903 - accuracy: 0.5840 - val_loss: 0.6451 - val_accuracy: 0.6456
Epoch 2/10
625/625 [==============================] - 68s 109ms/step - loss: 0.5862 - accuracy: 0.6982 - val_loss: 0.5845 - val_accuracy: 0.6730
Epoch 3/10
625/625 [==============================] - 69s 109ms/step - loss: 0.5060 - accuracy: 0.7578 - val_loss: 0.6268 - val_accuracy: 0.6554
Epoch 4/10
625/625 [==============================] - 65s 104ms/step - loss: 0.4331 - accuracy: 0.7990 - val_loss: 0.4581 - val_accuracy: 0.7812
Epoch 5/10
625/625 [==============================] - 68s 108ms/step - loss: 0.3640 - accuracy: 0.8350 - val_loss: 0.4941 - val_accuracy: 0.7758
Epoch 6/10
625/625 [==============================] - 66s 104ms/step - loss: 0.3077 - accuracy: 0.8626 - val_loss: 0.4698 - val_accuracy: 0.8162
Epoch 7/10
625/625 [==============================] - 65s 104ms/step - loss: 0.2392 - accuracy: 0.8985 - val_loss: 0.7119 - val_accuracy: 0.7224
Epoch 8/10
625/625 [==============================] - 68s 108ms/step - loss: 0.1875 - accuracy: 0.9251 - val_loss: 0.5163 - val_accuracy: 0.8174
Epoch 9/10
625/625 [==============================] - 66s 104ms/step - loss: 0.1431 - accuracy: 0.9420 - val_loss: 0.6315 - val_accuracy: 0.8266
Epoch 10/10
625/625 [==============================] - 65s 104ms/step - loss: 0.1184 - accuracy: 0.9545 - val_loss: 0.7637 - val_accuracy: 0.7912
```

```
import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'],color='red',label='train')
plt.plot(history.history['val_accuracy'],color='blue',label='validation')
plt.legend()
plt.show()
```

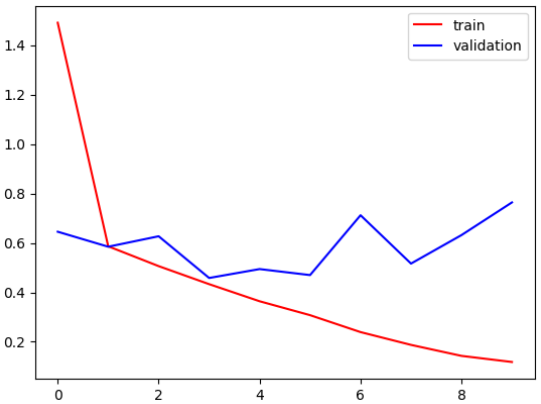
SAP ID:




```
plt.plot(history.history['accuracy'],color='red',label='train')
plt.plot(history.history['val_accuracy'],color='blue',label='validation')
plt.legend()
plt.show()
```



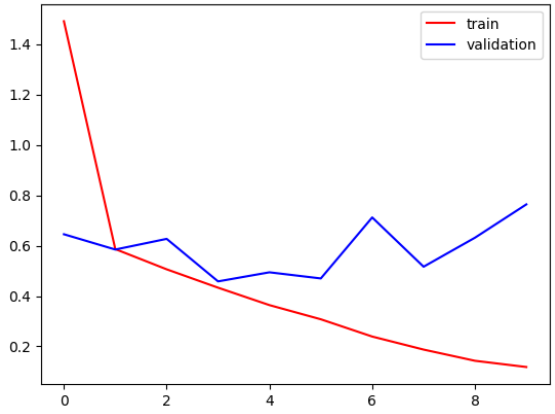

```
plt.plot(history.history['loss'],color='red',label='train')
plt.plot(history.history['val_loss'],color='blue',label='validation')
plt.legend()
plt.show()
```



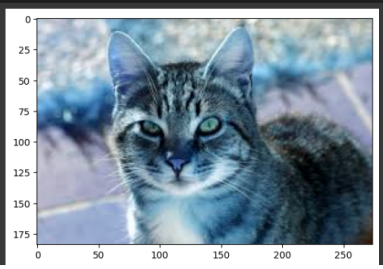

```
plt.plot(history.history['loss'],color='red',label='train')
plt.plot(history.history['val_loss'],color='blue',label='validation')
plt.legend()
plt.show()
```

SAP ID:

SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)




```
[16] import cv2

[25] from PIL import Image

[32] test_img = cv2.imread('/content/cat.jpeg')

[33] import numpy as np

[34] img_array = np.array(test_img)

[35] plt.imshow(img_array)
     plt.show()
```




```
[36] test_img.shape

     (184, 274, 3)

[37] test_img = cv2.resize(test_img,(256,256))

[38] test_input = test_img.reshape((1,256,256,3))

[39] model.predict(test_input)

     1/1 [==============================] - 0s 289ms/step
     array([[1.]], dtype=float32)
```