Experiment 7

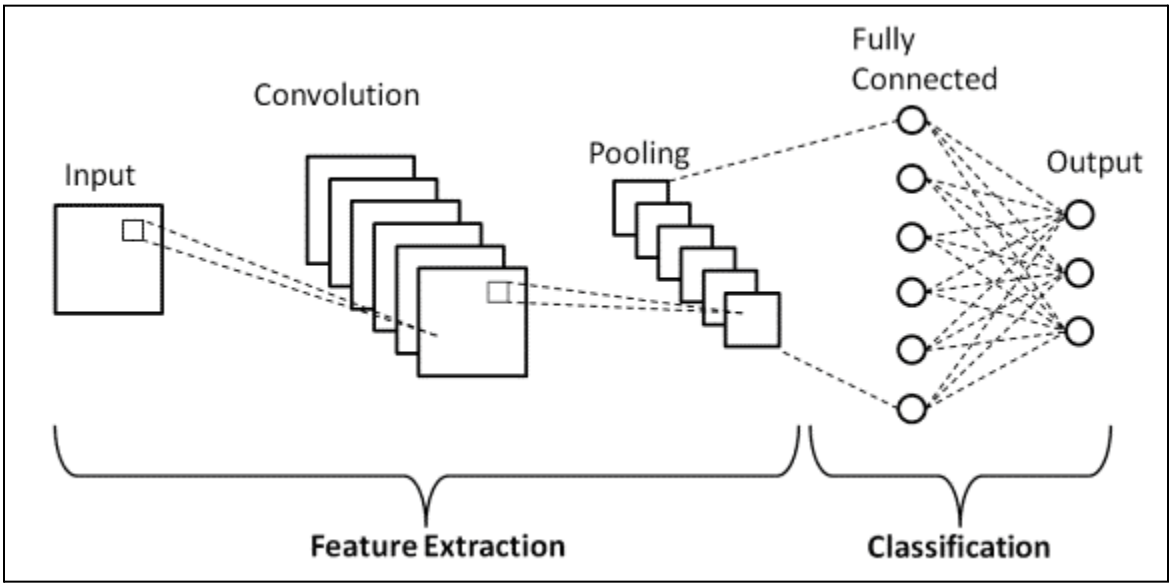
Aim - To implement CNN Deep Learning Applications like
i) Image Classification System
ii) Handwritten Digit Recognition System (like MNIST Dataset)
iii) Traffic Signs Recognition

Theory -

Convolutional Neural Networks (CNNs)

- Introduction to CNNs:
Convolutional Neural Networks (CNNs) are a class of deep learning models specifically designed for processing and classifying visual data, such as images and videos. They are inspired by the visual perception process in the human brain and have become the state-of-the-art technique for various computer vision tasks.

Key Components of a CNN:



1. Convolutional Layers: These layers apply convolutional operations to the input image. Convolutional filters (also known as kernels) slide over the input image to extract features like edges, textures, and patterns.

2. Pooling Layers: Pooling layers (e.g., Max Pooling) reduce the spatial dimensions of the feature maps, helping to make the network invariant to small changes in the input.

3. Fully Connected Layers: These layers connect every neuron in one layer to every neuron in the next layer, allowing the network to learn complex combinations of features.

- Working of a CNN:

1. Input: A CNN starts with an input image, which is passed through a series of convolutional and pooling layers. These layers extract hierarchical features from the input.

2. Flattening: The final convolutional or pooling layer is flattened into a vector.

3. Fully Connected Layers: The flattened vector is passed through one or more fully connected layers to make predictions. These layers learn to classify the input based on the extracted features.

4. Output: The output layer provides the final classification result, often using softmax activation for multi-class classification tasks.

- Building an Image Classification System for Dogs and Cats

- Application Overview:

In this experiment, we are developing an image classification system to distinguish between images of dogs and cats. This system is designed for binary classification, where the goal is to predict whether a given image contains a dog or a cat.

Code and Output -

- Model training code -

```
import tensorflow as tf
from keras.models import Sequential
```

```
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
import os

# Mount Google Drive if necessary
from google.colab import drive
drive.mount('/content/drive')

# Load and Preprocess the Dataset
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    validation_split=0.2
)

train_generator = train_datagen.flow_from_directory(
    '/content/drive/MyDrive/EXP7/train_set',
    target_size=(128, 128),
    batch_size=32,
    class_mode='binary',
    subset='training'
)

# Build the CNN Model
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128,
3)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
```

```python
# Train the Model
history = model.fit(
    train_generator,
    epochs=20,
)

# Print the Training Accuracy
train_accuracy = history.history['accuracy'][-1]
print(f'Training Accuracy: {train_accuracy}')

# Save the trained model for future use
model.save('/content/drive/MyDrive/EXP7/trained_model.h5')
```

- Output of training -

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True)
Found 32 images belonging to 2 classes.
Epoch 1/20
1/1 [==============================] - 3s 3s/step - loss: 0.6938 - accuracy: 0.5000
Epoch 2/20
1/1 [==============================] - 1s 1s/step - loss: 1.1547 - accuracy: 0.5000
Epoch 3/20
1/1 [==============================] - 1s 1s/step - loss: 0.6724 - accuracy: 0.5938
Epoch 4/20
1/1 [==============================] - 1s 1s/step - loss: 0.6943 - accuracy: 0.5000
Epoch 5/20
1/1 [==============================] - 1s 1s/step - loss: 0.6836 - accuracy: 0.5312
Epoch 6/20
1/1 [==============================] - 1s 1s/step - loss: 0.6772 - accuracy: 0.8125
Epoch 7/20
1/1 [==============================] - 1s 1s/step - loss: 0.6663 - accuracy: 0.5000
Epoch 8/20
1/1 [==============================] - 1s 1s/step - loss: 0.6618 - accuracy: 0.5000
Epoch 9/20
1/1 [==============================] - 2s 2s/step - loss: 0.6308 - accuracy: 0.7500
Epoch 10/20
1/1 [==============================] - 2s 2s/step - loss: 0.5981 - accuracy: 0.8125
Epoch 11/20
1/1 [==============================] - 1s 1s/step - loss: 0.5804 - accuracy: 0.6562
Epoch 12/20
1/1 [==============================] - 1s 1s/step - loss: 0.5585 - accuracy: 0.7812
Epoch 13/20
1/1 [==============================] - 1s 1s/step - loss: 0.5203 - accuracy: 0.8438
Epoch 14/20
1/1 [==============================] - 2s 2s/step - loss: 0.4874 - accuracy: 0.7188
Epoch 15/20
1/1 [==============================] - 1s 1s/step - loss: 0.4737 - accuracy: 0.8125
Epoch 16/20
1/1 [==============================] - 1s 1s/step - loss: 0.4027 - accuracy: 0.8750
Epoch 17/20
1/1 [==============================] - 1s 1s/step - loss: 0.4048 - accuracy: 0.8438
Epoch 18/20
1/1 [==============================] - 2s 2s/step - loss: 0.3239 - accuracy: 0.9062
Epoch 19/20
1/1 [==============================] - 2s 2s/step - loss: 0.2796 - accuracy: 0.8750
Epoch 20/20
1/1 [==============================] - 1s 1s/step - loss: 0.2277 - accuracy: 0.9688
```

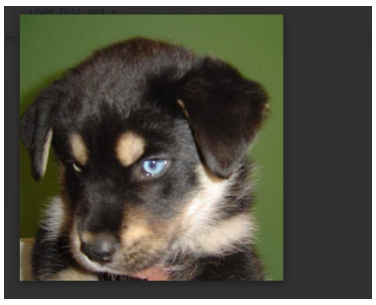
Accuracy -

```
Training Accuracy: 0.96875
```

- Trying out on a user given image code -

```
##Classiifying the image given by the user

import tensorflow as tf
from keras.preprocessing.image import load_img, img_to_array
import numpy as np

# Function to classify the user-uploaded image
def classify_image(image_path, model):
    img = load_img(image_path, target_size=(128, 128))
    img = img_to_array(img)
    img = np.expand_dims(img, axis=0)
    img = img / 255.0  # Normalize the image data

    prediction = model.predict(img)

    if prediction[0] > 0.5:
        return "It's a cat!"
    else:
        return "It's a dog!"

model =
tf.keras.models.load_model('/content/drive/MyDrive/EXP7/trained_model.h5')

# Example usage:
image_path = '/content/drive/MyDrive/EXP7/user_test_set/sample_image1.jpg'
result = classify_image(image_path, model)
print(result)
```

- User image -



- Output -

Conclusion -

Thus we have successfully implemented CNN model by building an image classification system to differentiate dogs and cats, with an accuracy of **96%.**