Program No. 4

AIM-Implement CNN to classify COVID-19 X-ray images in Python.

THEORY-

Convolutional Neural Networks (CNNs) are a powerful tool for machine learning, especially in tasks related to computer vision. Convolutional Neural Networks, or CNNs, are a specialized class of neural networks designed to effectively process grid-like data, such as images.

In this article, we are going to discuss convolutional neural networks (CNN) in machine learning in detail.

What is Convolutional Neural Network(CNN)?

A Convolutional Neural Network (CNN) is a type of deep learning algorithm that is particularly well-suited for image recognition and processing tasks. It is made up of multiple layers, including convolutional layers, pooling layers, and fully connected layers. The architecture of CNNs is inspired by the visual processing in the human brain, and they are well-suited for capturing hierarchical patterns and spatial dependencies within images.

Key components of a Convolutional Neural Network include:

1. **Convolutional Layers:** These layers apply convolutional operations to input images, using filters (also known as kernels) to detect features such as edges, textures, and more complex patterns. Convolutional operations help preserve the spatial relationships between pixels.
2. **Pooling Layers:** Pooling layers downsample the spatial dimensions of the input, reducing the computational complexity and the number of parameters in the network. Max pooling is a common pooling operation, selecting the maximum value from a group of neighboring pixels.
3. **Activation Functions:** Non-linear activation functions, such as Rectified Linear Unit (ReLU), introduce non-linearity to the model, allowing it to learn more complex relationships in the data.
4. **Fully Connected Layers:** These layers are responsible for making predictions based on the high-level features learned by the previous layers. They connect every neuron in one layer to every neuron in the next layer.

CNNs are trained using a large dataset of labeled images, where the network learns to recognize patterns and features that are associated with specific objects or classes. Proven to be highly effective in image-related tasks, achieving state-of-the-art performance in various computer vision applications. Their ability to automatically learn hierarchical representations of features makes them well-suited for tasks where the spatial relationships and patterns in the data are crucial for accurate predictions. CNNs are widely used in areas such as image classification, object detection, facial recognition, and medical image analysis.

The convolutional layers are the key component of a CNN, where filters are applied to the input image to extract features such as edges, textures, and shapes.

The output of the convolutional layers is then passed through pooling layers, which are used to down-sample the feature maps, reducing the spatial dimensions while retaining the most important information. The output of the pooling layers is then passed through one or more fully connected layers, which are used to make a prediction or classify the image.

Convolutional Neural Network Design

- The construction of a convolutional neural network is a multi-layered feed-forward neural network, made by assembling many unseen layers on top of each other in a particular order.
- It is the sequential design that give permission to CNN to learn hierarchical attributes.
- In CNN, some of them followed by grouping layers and hidden layers are typically convolutional layers followed by activation layers.
- The pre-processing needed in a ConvNet is kindred to that of the related pattern of neurons in the human brain and was motivated by the organization of the Visual Cortex.

Convolutional Neural Network Training

CNNs are trained using a supervised learning approach. This means that the CNN is given a set of labeled training images. The CNN then learns to map the input images to their correct labels.

The training process for a CNN involves the following steps:

1. **Data Preparation:** The training images are preprocessed to ensure that they are all in the same format and size.
2. **Loss Function:** A loss function is used to measure how well the CNN is performing on the training data. The loss function is typically calculated by taking the difference between the predicted labels and the actual labels of the training images.
3. **Optimizer:** An optimizer is used to update the weights of the CNN in order to minimize the loss function.
4. **Backpropagation:** Backpropagation is a technique used to calculate the gradients of the loss function with respect to the weights of the CNN. The gradients are then used to update the weights of the CNN using the optimizer.

CNN Evaluation

After training, CNN can be evaluated on a held-out test set. A collection of pictures that the CNN has not seen during training makes up the test set. How well the CNN performs on the test set is a good predictor of how well it will function on actual data.

The efficiency of a CNN on picture categorization tasks can be evaluated using a variety of criteria. Among the most popular metrics are:

- **Accuracy:** Accuracy is the percentage of test images that the CNN correctly classifies.
- **Precision:** Precision is the percentage of test images that the CNN predicts as a particular class and that are actually of that class.
- **Recall:** Recall is the percentage of test images that are of a particular class and that the CNN predicts as that class.

CODE and OUTPUT-

```
import tensorflow as tf

import matplotlib.pyplot as plt

from PIL import Image

tf.test.is_gpu_available()train_dir=r"archive (6)\Data\train"

test_dir=r"archive (6)/Data/test"

from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_data_generator=ImageDataGenerator(rescale=1./255,shear_range=0.2,zoom_range=0.2,horizontal_flip=True)

training_set=train_data_generator.flow_from_directory(r"archive (6)/Data/train",target_size=(64,64),batch_size=1,class_mode='categorical')
```

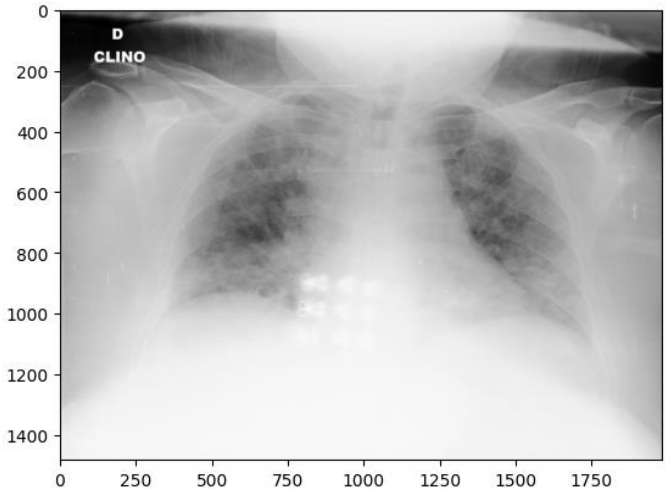
```
Found 5144 images belonging to 3 classes.
```

```
test_data_generator=ImageDataGenerator(rescale=1./255,)

testing_set=test_data_generator.flow_from_directory(r"archive (6)/Data/test",target_size=(64,64),batch_size=1,class_mode='categorical')
```

```
Found 1288 images belonging to 3 classes.
```

plt.imshow(Image.open(r"archive (6)/Data/train/COVID19/COVID19(1).jpg"))




```
from tensorflow.keras.utils import to_categorical

from tensorflow.keras.callbacks import EarlyStopping

from tensorflow.keras import models, layers

from tensorflow.keras.layers import *

model = models.Sequential()

model.add(layers.Conv2D(64, (3, 3), activation='relu',input_shape=(64,64,3)))

model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(64, (3, 3), activation='relu'))

model.add(layers.MaxPooling2D((2,2)))

 Flatten the 3D output to 1D

model.add(layers.Flatten())

 model.add(layers.Dense(64, activation='relu'))

model.add(layers.Dense(3, activation='sigmoid'))

early_stopping = EarlyStopping(patience=3)

model.compile(optimizer="adam", loss="categorical_crossentropy",
metrics=["accuracy"])

model.fit(x=training_set, validation_data=testing_set, epochs=5,
callbacks=[early_stopping])
```

```
Epoch 1/5
/home/arnav_bhatia/anaconda3/envs/py310/lib/python3.10/site-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/home/arnav_bhatia/anaconda3/envs/py310/lib/python3.10/site-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:122:
  self._warn_if_super_not_called()
5144/5144 ━━━━━━━━━━━━━━━━━━━━ 189s 36ms/step - accuracy: 0.7760 - loss: 0.5910 - val_accuracy: 0.9092 - val_loss: 0.2639
Epoch 2/5
5144/5144 ━━━━━━━━━━━━━━━━━━━━ 188s 36ms/step - accuracy: 0.8639 - loss: 0.3540 - val_accuracy: 0.9084 - val_loss: 0.2437
Epoch 3/5
5144/5144 ━━━━━━━━━━━━━━━━━━━━ 198s 38ms/step - accuracy: 0.8763 - loss: 0.3250 - val_accuracy: 0.9185 - val_loss: 0.2242
Epoch 4/5
5144/5144 ━━━━━━━━━━━━━━━━━━━━ 207s 40ms/step - accuracy: 0.8905 - loss: 0.2994 - val_accuracy: 0.9371 - val_loss: 0.1962
Epoch 5/5
5144/5144 ━━━━━━━━━━━━━━━━━━━━ 206s 40ms/step - accuracy: 0.9052 - loss: 0.2579 - val_accuracy: 0.9216 - val_loss: 0.2036
```

model.summary()

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 62, 62, 64)	1,792
max_pooling2d_2 (MaxPooling2D)	(None, 31, 31, 64)	0
conv2d_3 (Conv2D)	(None, 29, 29, 64)	36,928
max_pooling2d_3 (MaxPooling2D)	(None, 14, 14, 64)	0
flatten_1 (Flatten)	(None, 12544)	0
dense_1 (Dense)	(None, 3)	37,635

Total params: 229,067 (894.80 KB)

Trainable params: 76,355 (298.26 KB)

VIVA QUESTIONS-

1**. What is the ReLU activation function, and why is it used in CNNs?**

Answer:
ReLU (Rectified Linear Unit) is an activation function that introduces non-linearity into the network. It outputs the input directly if it is positive, otherwise, it outputs zero. It helps the network learn complex patterns.

2.Why are fully connected layers used at the end of CNNs?

Answer:
Fully connected layers (FC layers) at the end of CNNs take the high-level features extracted by convolutional layers and combine them to make final predictions. These layers allow for classification or regression tasks by outputting the final scores or probabilities.

3. What is the purpose of feature maps in CNN?

Answer:
Feature maps represent the output of a convolutional layer after applying filters to the input image. They highlight the presence of certain features (such as edges or textures) in different spatial regions of the image.

4. What is the difference between local and global features in CNN?

Answer:
Local features are extracted by convolutional layers and correspond to small patterns (e.g., edges, corners) in specific regions of the image. Global features, learned in deeper layers or fully connected layers, represent the overall structure or semantics of the image (e.g., object shapes).

5. How does CNN handle overfitting?

Answer:
Overfitting in CNN can be reduced through:

- **Data augmentation**: artificially increasing the size of the dataset.

- **Dropout**: randomly dropping neurons during training.
- **L2 regularization**: penalizing large weights.
- **Batch normalization**: normalizing the inputs to each layer.

6. What is transfer learning, and how is it applied to CNNs?

Answer:
Transfer learning involves taking a pre-trained CNN model (trained on a large dataset like ImageNet) and fine-tuning it for a different but related task by retraining only the final few layers or the entire network.

Program No. 3

AIM-Implementation of Convolutional Neural Network for MRI dataset in Python.

THEORY-

Introduction:

Convolutional Neural Networks (CNNs) are a specialized class of artificial neural networks that have proven to be extremely effective for processing and analyzing visual data, such as images or videos. They were inspired by the structure of the visual cortex in animals, where different neurons respond to specific local regions of the visual field.

CNNs have revolutionized computer vision tasks such as image classification, object detection, and face recognition, and are a fundamental part of modern deep learning architectures.

Basic Components of CNN:

1. Input Layer:

- The input to a CNN is typically an image represented as a matrix of pixel values. For a color image, there are usually three channels (RGB), so the input is a 3D matrix.

2. Convolutional Layer:

- The convolutional layer is the core building block of a CNN. This layer applies a set of learnable filters (or kernels) to the input data. Each filter slides over the input image (or the output of a previous layer) and performs element-wise multiplication followed by summation. This operation results in an output feature map.

- Key concepts:

- Filter/Kernel: A small matrix that is applied to the input data to detect specific features.

- Stride: Defines how much the filter moves across the input. A stride of 1 means the filter moves pixel by pixel, while a higher stride skips pixels.

- Padding: Adds extra pixels (typically zeros) around the border of the image. Padding ensures that the output feature map has the same spatial dimensions as the input.

3. Activation Function (ReLU):

- After the convolution operation, a non-linear activation function, typically the Rectified Linear Unit (ReLU), is applied to introduce non-linearity into the model. ReLU sets all negative values in the feature map to zero and allows positive values to pass

through, making it computationally efficient and helping prevent the vanishing gradient problem.

- ReLU Function: $f(x) = \max(0, x)$

4. Pooling Layer:

- Pooling is a down-sampling operation that reduces the spatial dimensions of the feature maps. This reduces the computational complexity and makes the network more robust to translations in the input.

- The two most common types of pooling are:

  - Max Pooling: Takes the maximum value from a patch of the feature map.

  - Average Pooling: Takes the average value from a patch of the feature map.

- Pooling layers reduce overfitting by compressing the information and making the representation more compact.

5. Fully Connected Layer (FC Layer):

- After several layers of convolution and pooling, the high-level reasoning in the network is done through fully connected layers. These layers take the flattened output from the previous layers and connect every neuron to every other neuron in the next layer.

- These layers are typically used in the final stages of CNNs to make predictions (e.g., for classification tasks).

6. Output Layer:

- The output layer depends on the task at hand. For a classification problem, the output layer often uses a softmax activation function to predict the probabilities of different classes.

- For regression tasks, the output might be a single or multiple continuous values.

1. Convolution Operation:

Given an input image I and a filter (kernel) K, the convolution operation is defined as:

$$
(I * K)(i, j) = \sum_{m}\sum_{n} I(i+m, j+n) \cdot K(m, n)
$$

\]

This operation slides the filter over the image and generates a feature map.

2. Pooling Operation:

For max pooling, given a region \(R \) of size \(f \times f \), the pooling function can be defined as:

\[

P(i, j) = \max_{(m,n) \in R(i,j)} I(m, n)

\]

The same applies to average pooling, but instead of the maximum, the average of the region values is computed.

3. Activation Function (ReLU):

\[

f(x) = \max(0, x)

\]

This is applied element-wise after each convolution operation.

Key Characteristics of CNNs:

1. Local Receptive Fields:

- CNNs exploit the local spatial structure of images. Filters are small and scan local regions (receptive fields) of the input image, capturing local patterns such as edges, textures, and corners.

- As we move deeper into the network, the receptive field increases, allowing the network to capture more global patterns and high-level features.

2. Weight Sharing:

- In fully connected networks, each neuron has its own set of weights. In CNNs, however, the same filter (set of weights) is applied across the entire input, which drastically reduces the number of parameters and computations.

- Weight sharing helps CNNs generalize better and reduces the risk of overfitting.

3. Translation Invariance:

- By using convolutional and pooling layers, CNNs become invariant to translations and small distortions in the input. For example, if an object in an image shifts slightly, CNNs can still recognize it due to the way features are extracted.

4. Hierarchical Feature Learning:

- CNNs learn hierarchical representations of images. The initial layers detect low-level features like edges, while deeper layers capture more complex patterns such as shapes and objects.

Training CNNs:

Training a CNN involves the following key steps:

1. Forward Propagation:

- The input image passes through several convolutional, activation, pooling, and fully connected layers, and the network outputs predictions based on the learned features.

2. Loss Function:

- The loss function (e.g., cross-entropy for classification tasks) calculates the difference between the predicted output and the actual labels.

3. Backpropagation:

- The gradients of the loss function with respect to the network parameters are computed using backpropagation. These gradients are then used to update the weights of the filters and fully connected layers via an optimization algorithm (like Stochastic Gradient Descent or Adam).

4. Optimization:

- The goal of training is to minimize the loss function. Optimization algorithms like Stochastic Gradient Descent (SGD) or Adam adjust the network's weights to minimize the prediction error.

Applications of CNNs:

1. Image Classification:

- CNNs are widely used for classifying images into predefined categories. Examples include facial recognition, object recognition, and medical imaging.

2. Object Detection:

- CNNs can be extended to detect objects within an image and place bounding boxes around them. Examples include YOLO (You Only Look Once) and Faster R-CNN.

3. Segmentation:

- In image segmentation, CNNs are used to label each pixel in an image as belonging to a particular class (e.g., background, road, building in an autonomous driving scenario).

4. Video Processing:

- CNNs can be used to analyze video frames for tasks such as action recognition, scene understanding, and video classification.

Conclusion:

CNNs have become the backbone of many computer vision applications due to their ability to automatically and efficiently capture important patterns in image data. By using layers of convolutions, pooling, and activations, CNNs build a hierarchy of features that allow them to perform tasks like image classification, object detection, and image segmentation with high accuracy.

CODE and OUTPUT-

```
import tensorflow as tf

import pandas as pd

import sklearn

from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_data_generator=ImageDataGenerator(rescale=1./255,shear_range=0.2,zoom_range=0.2,horizontal_flip=True)

training_set=train_data_generator.flow_from_directory(r"archive (6)/Data/train",target_size=(64,64),batch_size=32,class_mode='binary')

test_data_generator=ImageDataGenerator(rescale=1./255,)

testing_set=test_data_generator.flow_from_directory(r"archive (6)/Data/test",target_size=(64,64),batch_size=32,class_mode='binary')
```

```
Found 5144 images belonging to 3 classes.
Found 1288 images belonging to 3 classes.
```

```
from tensorflow.keras.callbacks import EarlyStopping
cnn=tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(filters=32,kernel_size=3,activation="relu",input_shape=[64,64,3
]),
    tf.keras.layers.MaxPool2D(strides=2,pool_size=2),
    tf.keras.layers.Conv2D(filters=32,kernel_size=3,activation="relu"),
    tf.keras.layers.MaxPool2D(strides=2,pool_size=2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(activation="relu",units=128),
    tf.keras.layers.Dense(activation="sigmoid",units=1)]

)
early_stopping=EarlyStopping(patience=3)
cnn.compile(optimizer="adam",loss="binary_crossentropy",metrics=["accuracy"])
cnn.fit(x=training_set,validation_data=testing_set,epochs=100,callbacks=[early_stopping
])

cnn.save("cnn1.h5")
```

```
  1/161 ━━━━━━━━━━━━━━━━━━━━ 26:01 10s/step - accuracy: 0.0000e+00 - loss: 0.7644
I0000 00:00:1726940168.451795   22580 device_compiler.h:188] Compiled cluster using XLA!  This line is logged at most once for th
161/161 ━━━━━━━━━━━━━━━━━━━━ 182s 1s/step - accuracy: 0.2399 - loss: -179219.6719 - val_accuracy: 0.2461 - val_loss: -6100921.000
Epoch 2/100
161/161 ━━━━━━━━━━━━━━━━━━━━ 180s 1s/step - accuracy: 0.2410 - loss: -24084588.0000 - val_accuracy: 0.2461 - val_loss: -159663152
Epoch 3/100
161/161 ━━━━━━━━━━━━━━━━━━━━ 183s 1s/step - accuracy: 0.2498 - loss: -294703840.0000 - val_accuracy: 0.2461 - val_loss: -96192038
Epoch 4/100
161/161 ━━━━━━━━━━━━━━━━━━━━ 187s 1s/step - accuracy: 0.2370 - loss: -1397114240.0000 - val_accuracy: 0.2461 - val_loss: -3200374
Epoch 5/100
161/161 ━━━━━━━━━━━━━━━━━━━━ 206s 1s/step - accuracy: 0.2479 - loss: -4346312192.0000 - val_accuracy: 0.2461 - val_loss: -7881203
Epoch 6/100
161/161 ━━━━━━━━━━━━━━━━━━━━ 211s 1s/step - accuracy: 0.2462 - loss: -9692965888.0000 - val_accuracy: 0.2461 - val_loss: -1616625
Epoch 7/100
161/161 ━━━━━━━━━━━━━━━━━━━━ 210s 1s/step - accuracy: 0.2435 - loss: -19712116736.0000 - val_accuracy: 0.2461 - val_loss: -292634
Epoch 8/100
161/161 ━━━━━━━━━━━━━━━━━━━━ 221s 1s/step - accuracy: 0.2485 - loss: -34770370560.0000 - val_accuracy: 0.2461 - val_loss: -482363
Epoch 9/100
```

VIVA QUESTIONS-

1. What is a Convolutional Neural Network (CNN)?

Answer:
A CNN is a class of deep learning models that is specifically designed to process and recognize patterns in visual data, such as images or videos. It automatically captures spatial hierarchies in data through convolutional layers, pooling layers, and fully connected layers.

2. What is the role of the convolutional layer in CNN?

Answer:
The convolutional layer applies filters (or kernels) to the input image or previous layer to extract features such as edges, textures, or other important characteristics by performing element-wise multiplications and summing the result.

3. What are filters in CNN, and how do they work?

Answer:
Filters (or kernels) are small matrices that slide across the input image, applying convolution operations. They help detect specific patterns (such as edges, textures) by capturing local spatial features of the input data.

4. What is stride in CNN?

Answer:
Stride refers to the number of pixels by which the filter shifts over the input image. A stride of 1 moves the filter by one pixel, while a stride of 2 skips one pixel for each movement, reducing the output size.

5. What is padding in CNN, and why is it used?

Answer:
Padding involves adding extra pixels (usually zeros) around the border of the input image to maintain the spatial dimensions after the convolution

operation. It ensures that the features at the edges of the image are also considered.

6. What is pooling, and what types of pooling are used in CNNs?

Answer:
Pooling is a down-sampling operation that reduces the spatial dimensions of the feature maps. The most common types are Max Pooling (which takes the maximum value) and Average Pooling (which takes the average value) within a region of the feature map.