Experiment No.3
Perform AI for medical diagnosis based on MRI/X-ray data using CNN
Date of Performance:
Date of Submission:

Aim: Perform AI for medical diagnosis based on MRI/X-ray data using CNN

Objective: The objective of this experiment is to use Convolutional Neural Networks to identify Pneumonia from the dataset of Xray images.

Theory:
Convolutional Neural Networks (CNNs), also known as ConvNets, are a class of deep learning neural networks specifically designed for tasks related to computer vision, such as image classification, object detection, image segmentation, and more. They have had a profound impact on various fields, including image and video analysis, medical imaging, autonomous vehicles, and many others. CNNs were inspired by the human visual system and are highly effective at processing grid-like data, making them particularly suited for tasks involving images and spatial data.

Here's an introduction to the key concepts and components of Convolutional Neural Networks:

Convolutional Layers (Convolution):

Convolution is the fundamental operation in CNNs. It involves sliding a small filter (also called a kernel) over the input data to perform element-wise multiplication and summation. This process helps detect features like edges, textures, and patterns within the data.

Convolutional layers consist of multiple filters that learn different features from the input data. These layers can capture increasingly complex features as you move deeper into the network.

Pooling Layers (Subsampling or Pooling):

Pooling layers are used to downsample the spatial dimensions of the feature maps produced by convolutional layers. Common pooling operations include max-pooling and average-pooling.

Pooling reduces the computational burden and helps make the network more robust to small translations and distortions in the input.

Activation Functions:

Activation functions like ReLU (Rectified Linear Unit) are applied to the output of convolutional and pooling layers. ReLU introduces non-linearity into the network, allowing it to learn complex relationships within the data.

Fully Connected Layers (FC Layers):

After several convolutional and pooling layers, CNNs often have one or more fully connected layers, which are traditional neural network layers. These layers perform high-level feature extraction and map the extracted features to the final output classes in tasks like image classification.

Flattening:

Before passing data from the convolutional layers to the fully connected layers, the feature maps are flattened into a one-dimensional vector. This transformation allows the network to treat the data as a traditional feedforward neural network.

Backpropagation:

CNNs are trained using backpropagation and gradient descent algorithms. During training, the network adjusts its internal parameters (weights and biases) to minimize a loss function, which measures the difference between the predicted output and the ground truth labels.

Architectural Variations:

CNN architectures can vary widely in terms of depth, number of layers, and specific components. Popular CNN architectures include LeNet, AlexNet, VGG, GoogLeNet (Inception), and ResNet, among others.
Transfer Learning:

CNNs have shown that pre-trained models on large datasets can be fine-tuned for specific tasks with smaller datasets. This approach, called transfer learning, has become common in computer vision applications.
CNNs have revolutionized the field of computer vision and are also used in various other domains like natural language processing and reinforcement learning. They excel at learning hierarchical representations of data, which makes them a powerful tool for a wide range of machine learning tasks involving structured grid-like data.

Code: -

```
[ ] from google.colab import drive
    drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
[ ] dataset_dir = '/content/drive/My Drive/dataset'
```

```
[ ] !ls '/content/drive/My Drive/dataset'
```

```
COVID  NORMAL
```

```
[ ] from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
[ ] # Define data generators
    train_datagen = ImageDataGenerator(
        rescale=1./255,
        rotation_range=20,
        zoom_range=0.2,
        shear_range=0.2,
        horizontal_flip=True,
        validation_split=0.2  # Reserve 20% for validation
    )
```

```
[ ] # Training set
    train_generator = train_datagen.flow_from_directory(
        dataset_dir,  # Path to the folder containing 'COVID' and 'NORMAL'
        target_size=(128, 128),  # Resize images
        batch_size=32,
        class_mode='binary',  # Binary classification (COVID or NORMAL)
        subset='training'  # Use this subset for training
    )
```

```
Found 260 images belonging to 2 classes.
```

```
# Validation set
validation_generator = train_datagen.flow_from_directory(
    dataset_dir,
    target_size=(128, 128),
    batch_size=32,
    class_mode='binary',
    subset='validation'  # Use this subset for validation
)
```

```
Found 64 images belonging to 2 classes.
```

```python
import tensorflow as tf

# Build CNN model
cnn_model = tf.keras.models.Sequential()

cnn_model.add(tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 3)))
cnn_model.add(tf.keras.layers.MaxPooling2D((2, 2)))
cnn_model.add(tf.keras.layers.Conv2D(64, (3, 3), activation='relu'))
cnn_model.add(tf.keras.layers.MaxPooling2D((2, 2)))
cnn_model.add(tf.keras.layers.Conv2D(128, (3, 3), activation='relu'))
cnn_model.add(tf.keras.layers.MaxPooling2D((2, 2)))

cnn_model.add(tf.keras.layers.Flatten())
cnn_model.add(tf.keras.layers.Dense(128, activation='relu'))
cnn_model.add(tf.keras.layers.Dropout(0.5))  # To prevent overfitting
cnn_model.add(tf.keras.layers.Dense(1, activation='sigmoid'))  # Binary output

# Compile the CNN model
cnn_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```python
# Train the CNN model
history = cnn_model.fit(
    train_generator,
    epochs=10,  # You can adjust the number of epochs
    validation_data=validation_generator
)
```

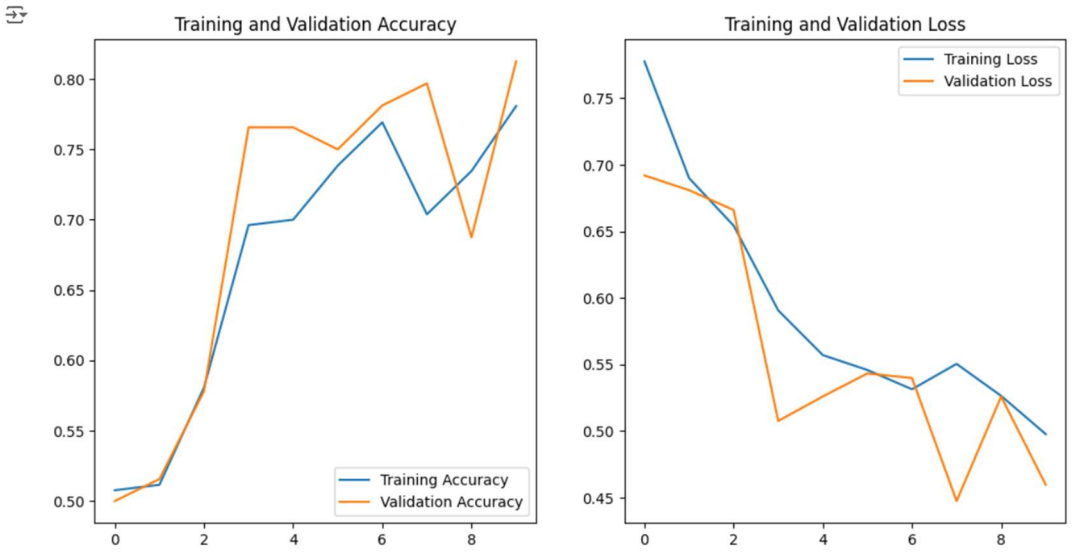
```
Epoch 1/10
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDa
  self._warn_if_super_not_called()
9/9 ━━━━━━━━━━━━━━━━━━━━ 97s 5s/step - accuracy: 0.5364 - loss: 0.8447 - val_accuracy: 0.5000 - val_loss: 0.6920
Epoch 2/10
9/9 ━━━━━━━━━━━━━━━━━━━━ 60s 1s/step - accuracy: 0.4374 - loss: 0.6954 - val_accuracy: 0.5156 - val_loss: 0.6810
Epoch 3/10
9/9 ━━━━━━━━━━━━━━━━━━━━ 13s 1s/step - accuracy: 0.5825 - loss: 0.6703 - val_accuracy: 0.5781 - val_loss: 0.6660
Epoch 4/10
9/9 ━━━━━━━━━━━━━━━━━━━━ 16s 2s/step - accuracy: 0.6557 - loss: 0.6166 - val_accuracy: 0.7656 - val_loss: 0.5076
Epoch 5/10
9/9 ━━━━━━━━━━━━━━━━━━━━ 13s 1s/step - accuracy: 0.6837 - loss: 0.5523 - val_accuracy: 0.7656 - val_loss: 0.5260
Epoch 6/10
9/9 ━━━━━━━━━━━━━━━━━━━━ 20s 1s/step - accuracy: 0.7387 - loss: 0.5437 - val_accuracy: 0.7500 - val_loss: 0.5433
Epoch 7/10
9/9 ━━━━━━━━━━━━━━━━━━━━ 21s 1s/step - accuracy: 0.7927 - loss: 0.5269 - val_accuracy: 0.7812 - val_loss: 0.5398
Epoch 8/10
9/9 ━━━━━━━━━━━━━━━━━━━━ 13s 1s/step - accuracy: 0.6815 - loss: 0.5626 - val_accuracy: 0.7969 - val_loss: 0.4474
Epoch 9/10
9/9 ━━━━━━━━━━━━━━━━━━━━ 21s 1s/step - accuracy: 0.7135 - loss: 0.5466 - val_accuracy: 0.6875 - val_loss: 0.5259
Epoch 10/10
9/9 ━━━━━━━━━━━━━━━━━━━━ 14s 1s/step - accuracy: 0.7551 - loss: 0.5218 - val_accuracy: 0.8125 - val_loss: 0.4596
```

```
# Plot accuracy and loss over time
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(10)

plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```



```
# Evaluate the model
test_loss, test_acc = cnn_model.evaluate(validation_generator)
print(f'Test Accuracy: {test_acc * 100:.2f}%')
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 341ms/step - accuracy: 0.7708 - loss: 0.4543
Test Accuracy: 76.56%
```

```
import numpy as np
from tensorflow.keras.preprocessing import image

# Load a new image for prediction
img_path = '/content/drive/My Drive/dataset/HSIB_Website_Report_Missed_det.fe260d5e.fill-10000x10000.jpg'
img = image.load_img(img_path, target_size=(128, 128))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0) / 255.0  # Rescale

# Predict
prediction = cnn_model.predict(img_array)
if prediction > 0.5:
    print("Prediction: NORMAL")
else:
    print("Prediction: COVID")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step
Prediction: COVID
```

Google Collaboratory Link: -

https://colab.research.google.com/drive/1UQFkwiCNAE0wTRu3YCaLtKQxCy6qsVMy?usp=sharing

Conclusion: -

Convolutional Neural Networks (CNNs) are well-suited for image data because they use convolutional layers to automatically detect spatial hierarchies of features, such as edges, textures, and shapes. This allows CNNs to capture local patterns and preserve spatial relationships in images. Additionally, weight sharing in convolutional layers reduces the number of parameters, making CNNs efficient and scalable for large image datasets. Pooling layers further help reduce

dimensionality while retaining important features, enhancing their effectiveness in image processing tasks.