Experiment No.3
Perform AI for medical diagnosis based on MRI/X-ray data using CNN
Date of Performance: 2/10/2023
Date of Submission: 9/10/2023

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.

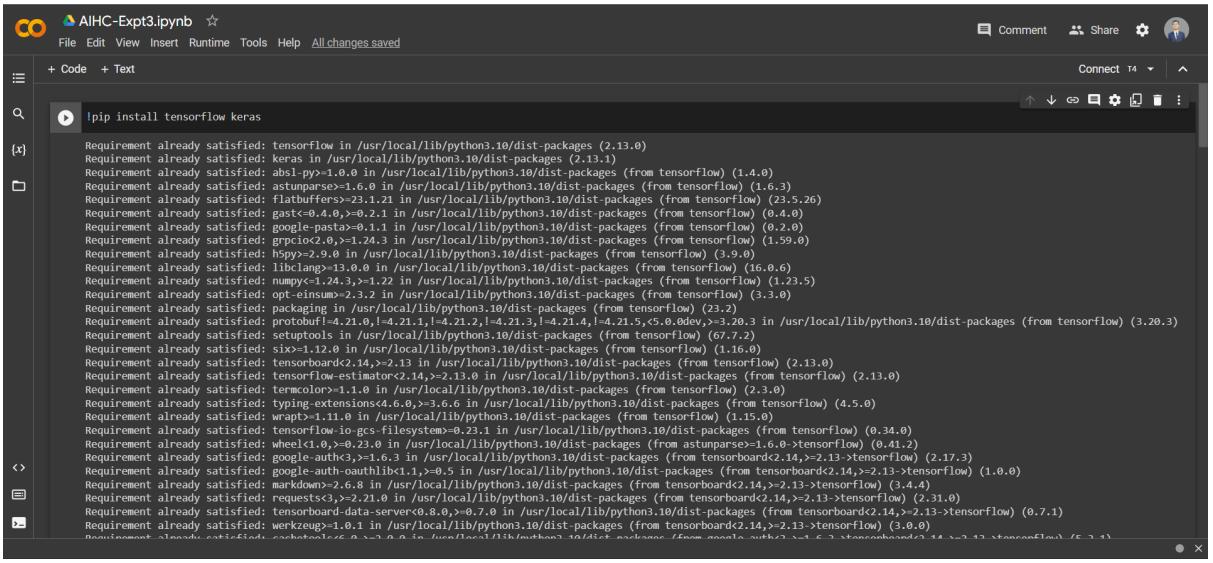
Backpropagation:
CNNs are trained using backpropagation and gradient descent algorithms. During training, the network adjusts its internal parameters (weights and biases) to minimize a loss function, which

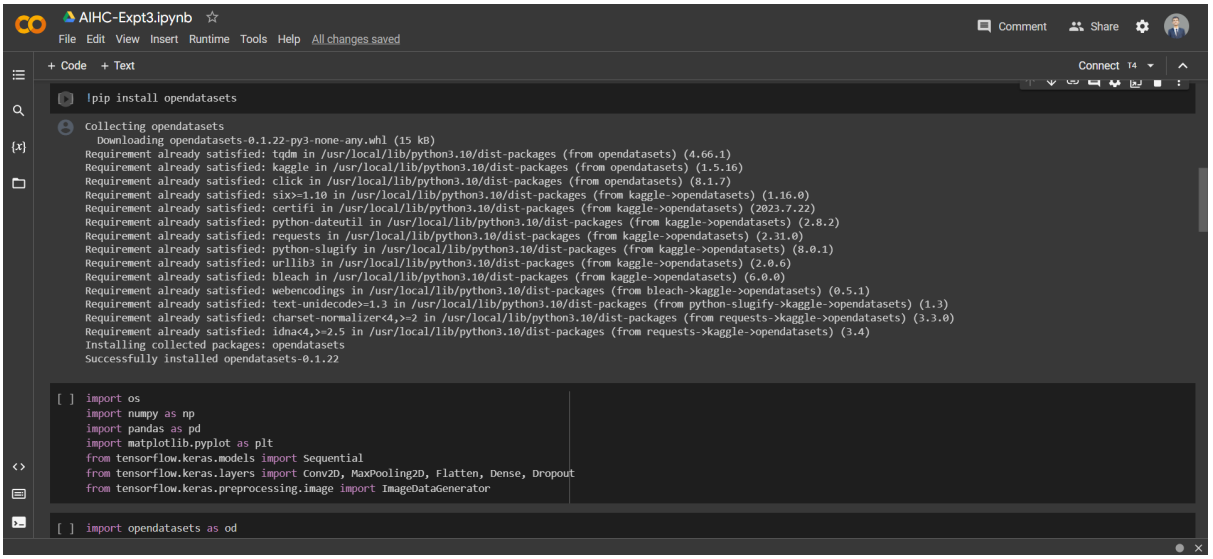
measures the difference between the predicted output and the ground truth labels.

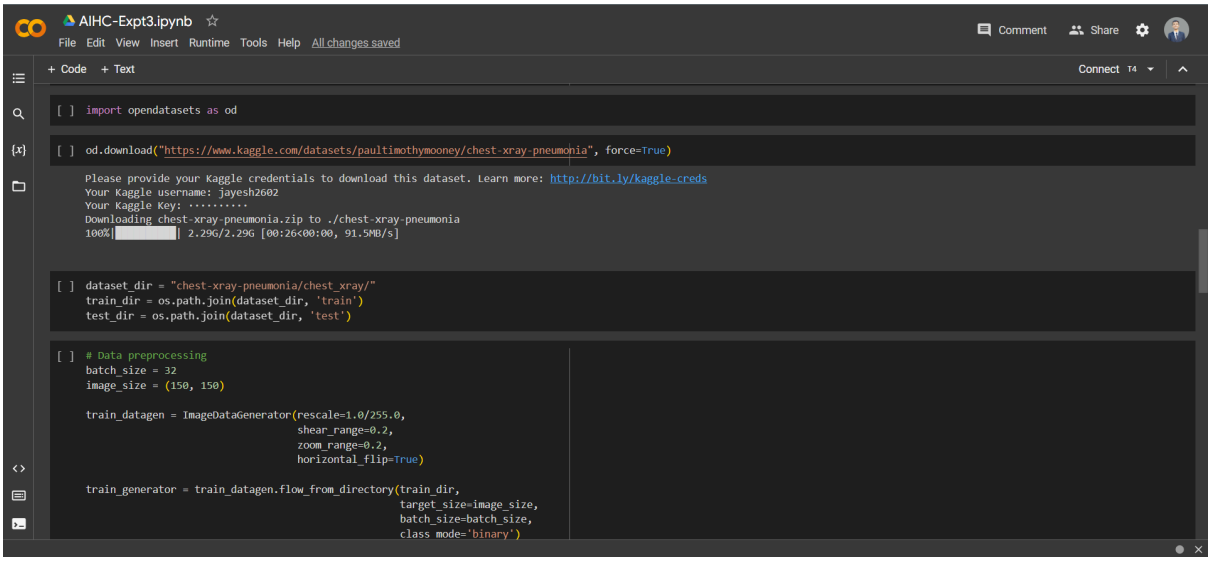
Architectural Variations:
CNN architectures can vary widely in terms of depth, number of layers, and specific components. Popular CNN architectures include LeNet, AlexNet, VGG, GoogLeNet (Inception), and ResNet, among others.

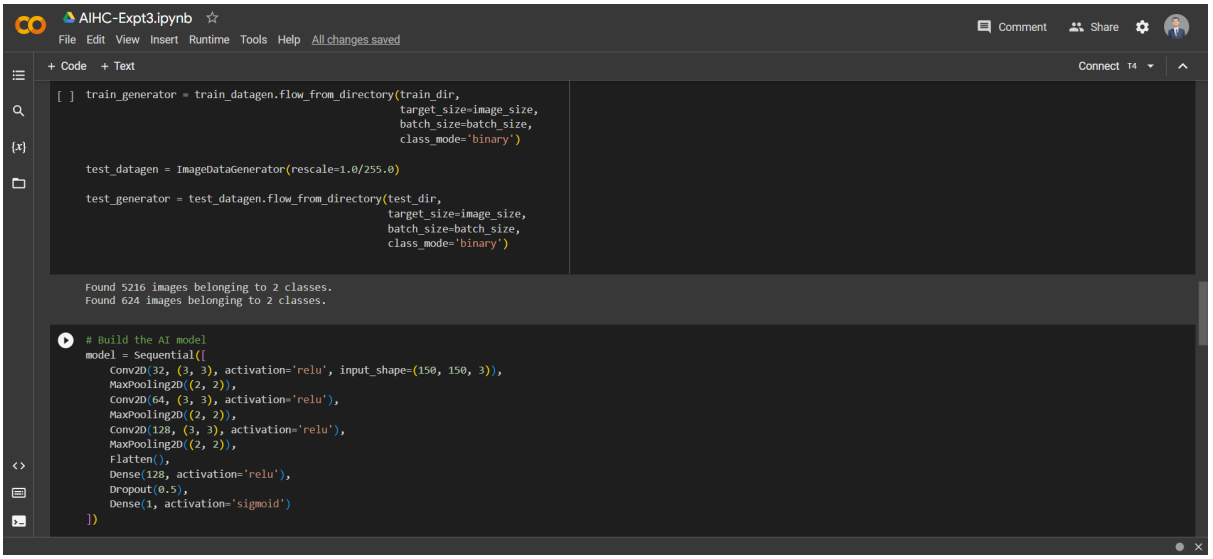
Transfer Learning:
CNNs have shown that pre-trained models on large datasets can be fine-tuned for specific tasks with smaller datasets. This approach, called transfer learning, has become common in computer vision applications.
CNNs have revolutionized the field of computer vision and are also used 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: -




AIHC-Expt3.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
Connect T4
!pip install opendatasets
Collecting opendatasets
Downloading opendatasets-0.1.22-py3-none-any.whl (15 kB)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from opendatasets) (4.66.1)
Requirement already satisfied: kaggle in /usr/local/lib/python3.10/dist-packages (from opendatasets) (1.5.16)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from opendatasets) (8.1.7)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.10/dist-packages (from kaggle->opendatasets) (1.16.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages (from kaggle->opendatasets) (2023.7.22)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-packages (from kaggle->opendatasets) (2.8.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from kaggle->opendatasets) (2.31.0)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.10/dist-packages (from kaggle->opendatasets) (8.0.1)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages (from kaggle->opendatasets) (2.0.6)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from kaggle->opendatasets) (6.0.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from bleach->kaggle->opendatasets) (0.5.1)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.10/dist-packages (from python-slugify->kaggle->opendatasets) (1.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle->opendatasets) (3.3.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle->opendatasets) (3.4)
Installing collected packages: opendatasets
Successfully installed opendatasets-0.1.22
[ ] import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator
[ ] import opendatasets as od


AIHC-Expt3.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
Connect T4
[ ] import opendatasets as od
[ ] od.download("https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia", force=True)
Please provide your Kaggle credentials to download this dataset. Learn more: http://bit.ly/kaggle-creds
Your Kaggle username: jayesh2602
Your Kaggle Key: ·····
Downloading chest-xray-pneumonia.zip to ./chest-xray-pneumonia
100%| | 2.29G/2.29G [00:26<00:00, 91.5MB/s]
[ ] dataset_dir = "chest-xray-pneumonia/chest_xray/"
train_dir = os.path.join(dataset_dir, 'train')
test_dir = os.path.join(dataset_dir, 'test')
[ ] # Data preprocessing
batch_size = 32
image_size = (150, 150)
train_datagen = ImageDataGenerator(rescale=1.0/255.0,
shear_range=0.2,
zoom_range=0.2,
horizontal_flip=True)
train_generator = train_datagen.flow_from_directory(train_dir,
target_size=image_size,
batch_size=batch_size,
class_mode='binary')



AIHC-Expt3.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

Connect T4

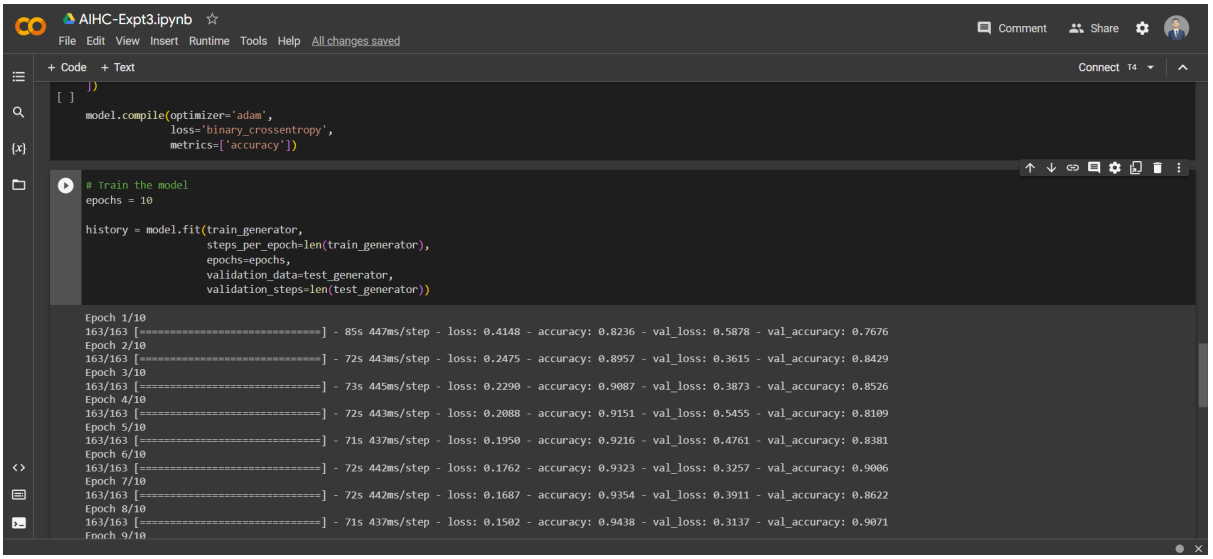
```
[ ] train_generator = train_datagen.flow_from_directory(train_dir,
                                                        target_size=image_size,
                                                        batch_size=batch_size,
                                                        class_mode='binary')

    test_datagen = ImageDataGenerator(rescale=1.0/255.0)

    test_generator = test_datagen.flow_from_directory(test_dir,
                                                      target_size=image_size,
                                                      batch_size=batch_size,
                                                      class_mode='binary')
```

```
Found 5216 images belonging to 2 classes.
Found 624 images belonging to 2 classes.
```

```
# Build the AI model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])
```



AIHC-Expt3.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

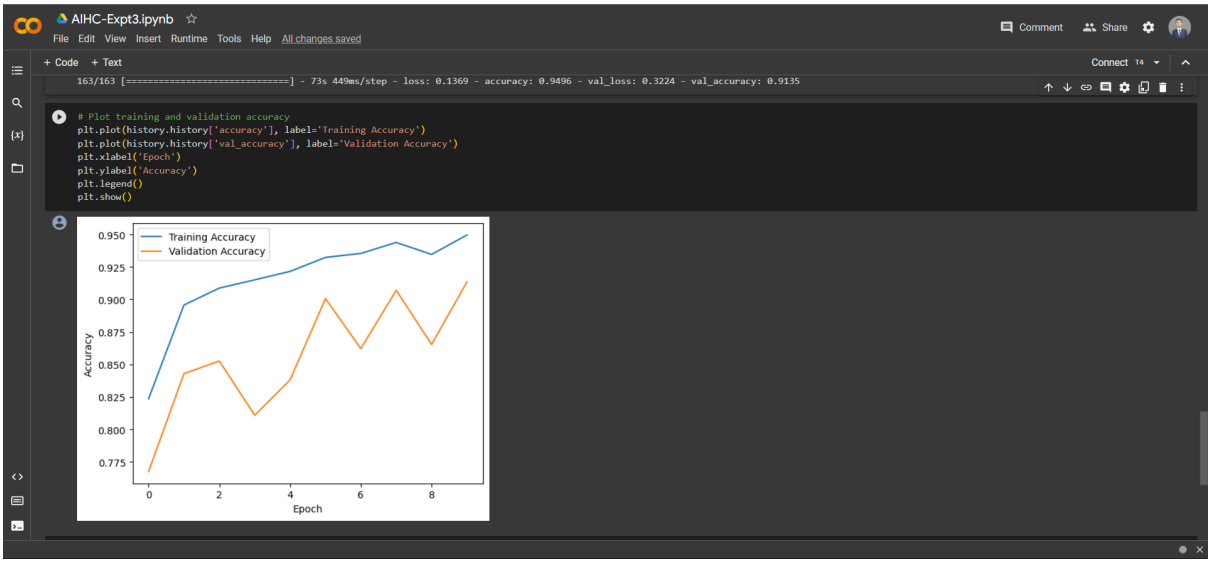
Connect T4

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

```
# Train the model
epochs = 10

history = model.fit(train_generator,
                    steps_per_epoch=len(train_generator),
                    epochs=epochs,
                    validation_data=test_generator,
                    validation_steps=len(test_generator))
```

```
Epoch 1/10
163/163 [==============================] - 85s 447ms/step - loss: 0.4148 - accuracy: 0.8236 - val_loss: 0.5878 - val_accuracy: 0.7676
Epoch 2/10
163/163 [==============================] - 72s 443ms/step - loss: 0.2475 - accuracy: 0.8957 - val_loss: 0.3615 - val_accuracy: 0.8429
Epoch 3/10
163/163 [==============================] - 73s 445ms/step - loss: 0.2290 - accuracy: 0.9087 - val_loss: 0.3873 - val_accuracy: 0.8526
Epoch 4/10
163/163 [==============================] - 72s 443ms/step - loss: 0.2088 - accuracy: 0.9151 - val_loss: 0.5455 - val_accuracy: 0.8109
Epoch 5/10
163/163 [==============================] - 71s 437ms/step - loss: 0.1950 - accuracy: 0.9216 - val_loss: 0.4761 - val_accuracy: 0.8381
Epoch 6/10
163/163 [==============================] - 72s 442ms/step - loss: 0.1762 - accuracy: 0.9323 - val_loss: 0.3257 - val_accuracy: 0.9006
Epoch 7/10
163/163 [==============================] - 72s 442ms/step - loss: 0.1687 - accuracy: 0.9354 - val_loss: 0.3911 - val_accuracy: 0.8622
Epoch 8/10
163/163 [==============================] - 71s 437ms/step - loss: 0.1502 - accuracy: 0.9438 - val_loss: 0.3137 - val_accuracy: 0.9071
Epoch 9/10
```



Google Collaboratory Link: -

AIHC-Expt3.ipynb

Conclusion: -

CNNs are better suited for large datasets and challenging image classification tasks than ANNs since they are more efficient and scalable. Because CNNs require a lot of training data and are computationally costly, they may not be suitable for many applications.