**EXPERIMENT 7**

**OBJECTIVE :**  WAP to retrain a pretrained imagenet model to classify a medical image dataset.

**DESCRIPTION OF MODEL**

This model utilizes the VGG16 architecture pre-trained on ImageNet for transfer learning.

The pre-trained VGG16 model is used as a base model with the convolutional layers frozen.

Data augmentation to increase the diversity of the training data and avoid overfitting on small datasets.

Custom fully connected layers are added on top to classify the images based on the provided dataset.

1st Dense layer with 256 neurons followed by dropout 0.5 and output layer with 1 neuron sigmoid activation function .

The model uses a global average pooling layer to reduce the dimensions before passing the data to the fully connected layers.

The model is trained with binary crossentropy loss and an Adam optimizer.

The final classification layer is a dense layer with a sigmoid activation function for binary classification (COVID vs Non-COVID).

1. Mount Google Drive

from google.colab import drive

drive.mount('/content/drive')

2. Upload ZIP from Local System

from google.colab import files

import zipfile, os

uploaded = files.upload()  # Upload COVID\_CT\_dataset.zip

for fn in uploaded.keys():

    zip\_path = fn

    extract\_dir = '/content/covid\_data'

    with zipfile.ZipFile(zip\_path, 'r') as zip\_ref:

        zip\_ref.extractall(extract\_dir)

3. Imports

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import f1\_score, accuracy\_score, roc\_auc\_score, classification\_report, roc\_curve

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.applications import VGG16

from tensorflow.keras.models import Model

from tensorflow.keras.layers import Flatten, Dense, Dropout

from tensorflow.keras.optimizers import Adam

4. Dataset Setup

data\_dir = os.path.join(extract\_dir, 'COVID\_CT\_dataset')

categories = ['CT\_COVID', 'CT\_NonCOVID']

filepaths, labels = [], []

for category in categories:

    folder = os.path.join(data\_dir, category)

    for fname in os.listdir(folder):

        if fname.lower().endswith(('.png', '.jpg', '.jpeg')):

            filepaths.append(os.path.join(folder, fname))

            labels.append(category)

df = pd.DataFrame({'filename': filepaths, 'label': labels})

5. Train-Test Split

train\_df, test\_df = train\_test\_split(df, test\_size=0.2, stratify=df['label'], random\_state=42)

6. Data Generators

img\_size = 224

batch\_size = 32

datagen = ImageDataGenerator(rescale=1./255)

train\_gen = datagen.flow\_from\_dataframe(

    train\_df, x\_col='filename', y\_col='label',

    target\_size=(img\_size, img\_size), batch\_size=batch\_size,

    class\_mode='binary'

)

test\_gen = datagen.flow\_from\_dataframe(

    test\_df, x\_col='filename', y\_col='label',

    target\_size=(img\_size, img\_size), batch\_size=1,

    class\_mode='binary', shuffle=False

)

7. Build Model

base\_model = VGG16(weights='imagenet', include\_top=False, input\_shape=(img\_size, img\_size, 3))

base\_model.trainable = False

x = Flatten()(base\_model.output)

x = Dense(256, activation='relu')(x)

x = Dropout(0.5)(x)

output = Dense(1, activation='sigmoid')(x)

model = Model(inputs=base\_model.input, outputs=output)

model.compile(optimizer=Adam(1e-4), loss='binary\_crossentropy', metrics=['accuracy'])

8. Train Model

history = model.fit(train\_gen, epochs=10)

9. Evaluate Model

pred\_probs = model.predict(test\_gen)

preds = (pred\_probs > 0.5).astype(int).reshape(-1)

true = test\_gen.classes

f1 = f1\_score(true, preds)

acc = accuracy\_score(true, preds)

auc = roc\_auc\_score(true, pred\_probs)

report = classification\_report(true, preds, target\_names=test\_gen.class\_indices.keys())

print("\nClassification Report:\n")

print(report)

print(f"F1 Score: {f1:.4f}")

print(f"Accuracy: {acc:.4f}")

print(f"AUC: {auc:.4f}")

10. Save Results to Google Drive

results\_path = '/content/drive/MyDrive/nnexp7\_files'

os.makedirs(results\_path, exist\_ok=True)

(a) Save metrics

with open(f'{results\_path}/metrics\_report.txt', 'w') as f:

    f.write("Classification Report:\n")

    f.write(report + '\n')

    f.write(f"F1 Score: {f1:.4f}\n")

    f.write(f"Accuracy: {acc:.4f}\n")

    f.write(f"AUC: {auc:.4f}\n")

(b) Save model

model.save(f'{results\_path}/vgg16\_covid\_classifier.h5')

(c) Plot accuracy/loss curves

plt.figure(figsize=(10,4))

plt.subplot(1,2,1)

plt.plot(history.history['accuracy'], label='Train Accuracy')

plt.title('Accuracy Curve')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.subplot(1,2,2)

plt.plot(history.history['loss'], label='Train Loss', color='orange')

plt.title('Loss Curve')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.tight\_layout()

plt.savefig(f'{results\_path}/training\_curves.png')

plt.show()

(d) Plot AUC curve

fpr, tpr, \_ = roc\_curve(true, pred\_probs)

plt.figure()

plt.plot(fpr, tpr, label=f"AUC = {auc:.4f}")

plt.plot([0, 1], [0, 1], linestyle='--', color='gray')

plt.title('ROC Curve')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.legend(loc='lower right')

plt.savefig(f'{results\_path}/auc\_curve.png')

print(f"\n All results saved to: {results\_path}")

**OUTPUT**

Epoch 10/10

**19/19** ━━━━━━━━━━━━━━━━━━━━ **3s** 153ms/step - accuracy: 0.9721 - loss: 0.1130

Classification Report:

precision recall f1-score support

CT\_COVID 0.91 0.83 0.87 70

CT\_NonCOVID 0.86 0.93 0.89 80

accuracy 0.88 150

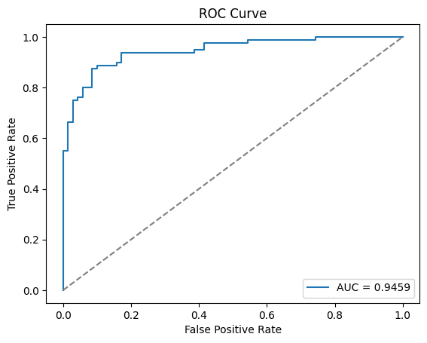
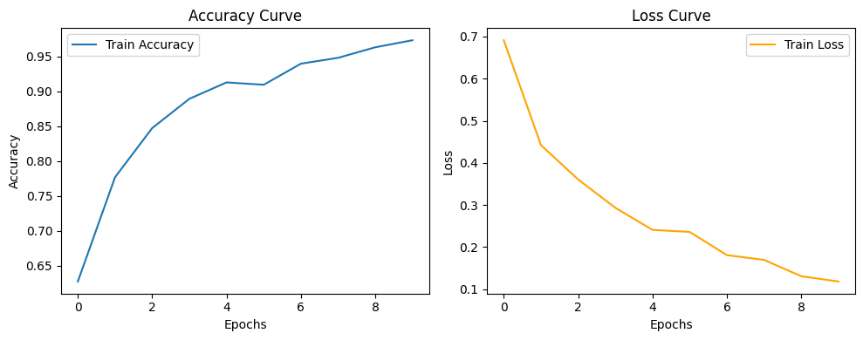
macro avg 0.88 0.88 0.88 150

weighted avg 0.88 0.88 0.88 150

F1 Score: 0.8916

Accuracy: 0.8800

AUC: 0.9459



"""

**Description of Code:**

1. The model uses a pre-trained VGG16 as the base, keeping its layers frozen to leverage its feature extraction capabilities.

2. The base VGG16 model is followed by fully connected layers.

3. Fully connected layers (Dense layers) are added with ReLU activations to learn higher-level representations.

4. Dropout is applied to prevent overfitting and improve generalization.

5. The model ends with a sigmoid output layer for binary classification.

6. The model is trained with binary crossentropy loss and an Adam optimizer.

7. Performance is evaluated using train los and accuracy curves and f1 score , confusion matrix , area under curve .

Classification Report:

              precision    recall  f1-score   support

    CT\_COVID       0.91      0.83      0.87        70

 CT\_NonCOVID       0.86      0.93      0.89        80

    accuracy                           0.88       150

   macro avg       0.88      0.88      0.88       150

weighted avg       0.88      0.88      0.88       150

F1 Score: 0.8916

Accuracy: 0.8800

AUC: 0.9459

PERFORMANCE EVLUATION

After optimising for 6 cases of fully connected layers , maximum accuracy achieved is 88.00 % .

**MY COMMENTS**

Limitations:

- The maximum accuracy achieved was 88%, which might not be sufficient.

- Increasing the number of Dense layers significantly decreased accuracy.

BatchNormalization and Global Average Pooling did not improve accuracy .

The model is using VGG16 pre-trained weights, which may not generalize well to specific datasets, especially those that differ from the ImageNet dataset.

# Scope for Improvement:

Accuracy can be improved by

# 1. Fine-tuning VGG16 instead of freezing all layers in the VGG16 model.

# 2. Hyperparameter Tuning: The batch size, learning rate, and number of Dense layers could be tuned for better performance.

# 3. More Advanced Architectures ResNet or EfficientNet .s.

# 3. Regularization: Exploring different regularization methods (e.g., L2 regularization) .

# 5. Learning Rate Scheduling: Implementing learning rate decay or using a learning rate scheduler (e.g., ReduceLROnPlateau) .

"""