**AI BASED LIVER CANCER DETECTION USING**

**RESIDUAL NETWORK ARCHITECTURE**

**TRAINING THE SYSTEM WITH THE DATASET**

y\_train=lb.transform(y\_train)

y\_train1 = to\_categorical(y\_train)

#y\_test=lb.transform(y\_test)

print(len(X\_train))

print(len(y\_train))

#Moulding train images

train\_datagen = image.ImageDataGenerator(rescale = 1./255, shear\_range = 0.2,zoom\_range = 0.2, horizontal\_flip = True)

test\_dataset = image.ImageDataGenerator()

#Reshaping test and validation images

train\_generator = train\_datagen.flow\_from\_directory(

'/content/drive/MyDrive/liver dataset/train',

target\_size = (224,224),

batch\_size = 10,

class\_mode = 'binary')

validation\_generator = test\_dataset.flow\_from\_directory(

'/content/drive/MyDrive/liver dataset/val',

target\_size = (224,224),

batch\_size = 10,

class\_mode = 'binary')

def identity\_block(X, f, filters, stage, block):

conv\_name\_base = 'res' + str(stage) + block + '\_branch'

bn\_name\_base = 'bn' + str(stage) + block + '\_branch'

F1, F2, F3 = filters

X\_shortcut = X

X = Conv2D(filters=F1, kernel\_size=(1, 1), strides=(1, 1), padding='valid', name=conv\_name\_base + '2a', kernel\_initializer=glorot\_uniform(seed=0))(X)

X = BatchNormalization(axis=3, name=bn\_name\_base + '2a')(X)

X = Activation('relu')(X)

X = Conv2D(filters=F2, kernel\_size=(f, f), strides=(1, 1), padding='same', name=conv\_name\_base + '2b', kernel\_initializer=glorot\_uniform(seed=0))(X)

X = BatchNormalization(axis=3, name=bn\_name\_base + '2b')(X)

X = Activation('relu')(X)

X = Conv2D(filters=F3, kernel\_size=(1, 1), strides=(1, 1), padding='valid', name=conv\_name\_base + '2c', kernel\_initializer=glorot\_uniform(seed=0))(X)

X = BatchNormalization(axis=3, name=bn\_name\_base + '2c')(X)

X = Add()([X, X\_shortcut])# SKIP Connection

X = Activation('relu')(X)

return X

def convolutional\_block(X, f, filters, stage, block, s=2):

conv\_name\_base = 'res' + str(stage) + block + '\_branch'

bn\_name\_base = 'bn' + str(stage) + block + '\_branch'

F1, F2, F3 = filters

X\_shortcut = X

X = Conv2D(filters=F1, kernel\_size=(1, 1), strides=(s, s), padding='valid', name=conv\_name\_base + '2a', kernel\_initializer=glorot\_uniform(seed=0))(X)

X = BatchNormalization(axis=3, name=bn\_name\_base + '2a')(X)

X = Activation('relu')(X)

X = Conv2D(filters=F2, kernel\_size=(f, f), strides=(1, 1), padding='same', name=conv\_name\_base + '2b', kernel\_initializer=glorot\_uniform(seed=0))(X)

X = BatchNormalization(axis=3, name=bn\_name\_base + '2b')(X)

X = Activation('relu')(X)

X = Conv2D(filters=F3, kernel\_size=(1, 1), strides=(1, 1), padding='valid', name=conv\_name\_base + '2c', kernel\_initializer=glorot\_uniform(seed=0))(X)

X = BatchNormalization(axis=3, name=bn\_name\_base + '2c')(X)

X\_shortcut = Conv2D(filters=F3, kernel\_size=(1, 1), strides=(s, s), padding='valid', name=conv\_name\_base + '1', kernel\_initializer=glorot\_uniform(seed=0))(X\_shortcut)

X\_shortcut = BatchNormalization(axis=3, name=bn\_name\_base + '1')(X\_shortcut)

X = Add()([X, X\_shortcut])

X = Activation('relu')(X)

return X

def ResNet50(input\_shape=(224, 224, 3)):

X\_input = Input(input\_shape)

X = ZeroPadding2D((3, 3))(X\_input)

X = Conv2D(64, (7, 7), strides=(2, 2), name='conv1', kernel\_initializer=glorot\_uniform(seed=0))(X)

X = BatchNormalization(axis=3, name='bn\_conv1')(X)

X = Activation('relu')(X)

X = MaxPooling2D((3, 3), strides=(2, 2))(X)

X = convolutional\_block(X, f=3, filters=[64, 64, 256], stage=2, block='a', s=1)

X = identity\_block(X, 3, [64, 64, 256], stage=2, block='b')

X = identity\_block(X, 3, [64, 64, 256], stage=2, block='c')

X = convolutional\_block(X, f=3, filters=[128, 128, 512], stage=3, block='a', s=2)

X = identity\_block(X, 3, [128, 128, 512], stage=3, block='b')

X = identity\_block(X, 3, [128, 128, 512], stage=3, block='c')

X = identity\_block(X, 3, [128, 128, 512], stage=3, block='d')

X = convolutional\_block(X, f=3, filters=[256, 256, 1024], stage=4, block='a', s=2)

X = identity\_block(X, 3, [256, 256, 1024], stage=4, block='b')

X = identity\_block(X, 3, [256, 256, 1024], stage=4, block='c')

X = identity\_block(X, 3, [256, 256, 1024], stage=4, block='d')

X = identity\_block(X, 3, [256, 256, 1024], stage=4, block='e')

X = identity\_block(X, 3, [256, 256, 1024], stage=4, block='f')

X = X = convolutional\_block(X, f=3, filters=[512, 512, 2048], stage=5, block='a', s=2)

X = identity\_block(X, 3, [512, 512, 2048], stage=5, block='b')

X = identity\_block(X, 3, [512, 512, 2048], stage=5, block='c')

X = AveragePooling2D(pool\_size=(2, 2), padding='same')(X)

model = Model(inputs=X\_input, outputs=X, name='ResNet50')

return model

base\_model = ResNet50(input\_shape=(224, 224, 3))

from tensorflow.keras.layers import Bidirectional

from tensorflow.keras.layers import LSTM

headModel = base\_model.output

headModel = Flatten()(headModel)

headModel=Dense(64, activation='relu', name='fc1',kernel\_initializer=glorot\_uniform(seed=0))(headModel)

headModel=Dense(128, activation='relu', name='fc2',kernel\_initializer=glorot\_uniform(seed=0))(headModel)

headModel = Dense(1,activation='sigmoid', name='fc3',kernel\_initializer=glorot\_uniform(seed=0))(headModel)

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

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

base\_model.load\_weights("/content/drive/MyDrive/resnet50\_weights\_tf\_dim\_ordering\_tf\_kernels\_notop.h5")

for layer in base\_model.layers:

layer.trainable = False

for layer in model.layers:

print(layer, layer.trainable)

es=EarlyStopping(monitor='val\_accuracy', mode='max', verbose=1, patience=30)

mc = ModelCheckpoint('model.h5', monitor='val\_accuracy')

print(y\_train)

H = model.fit\_generator(train\_generator,validation\_data=validation\_generator,epochs=15,verbose=1,callbacks=[mc,es])

H = model.fit\_generator(train\_generator,validation\_data=validation\_generator,epochs=15,verbose=1,callbacks=[mc,es])

import matplotlib.pyplot as plt

plt.figure(figsize=(20,10))

plt.subplot(1, 2, 1)

plt.suptitle('Optimizer : ADAM ', fontsize=10)

plt.ylabel('Loss', fontsize=16)

plt.plot(H.history['loss'], label='Training Loss')

plt.legend(loc='upper right')

plt.subplot(1, 2, 2)

plt.ylabel('Accuracy', fontsize=16)

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

plt.legend(loc='lower right')

plt.show()

image.png