**DETECTION OF INTRACRANIAL BLEEDING USING AN EFFECTIVE NEURAL NETWORK**

**Software tool used:**

* **IDE –** Anaconda Navigator
* **Programming Language –** Python

**Executable Code: [Alexnet]**

import os

import matplotlib.pyplot as plt

import seaborn as sn

import cv2

from random import randint

import numpy as np

CLASSES, bleed = [], [] # names of classes, count of images for each class

for root, dirs, files in os.walk('input/images'):

f = os.path.basename(root) # get class name

if len(files) > 0:

bleed.append(len(files))

if f not in CLASSES:

CLASSES.append(f) # add folder name

ax.legend(fontsize = 12);

bleed\_count = len(CLASSES)

print('{} classes with {} images in total'.format(len(CLASSES), sum(bleed))

img\_w, img\_h = 227, 227 # width and height of image

train\_dir = 'input/images/train/'

def read\_imgs\_lbls(\_dir):

Images, Labels = [], []

for root, dirs, files in os.walk(\_dir):

f = os.path.basename(root) # get class name

for file in files:

Labels.append(f)

try:

image = cv2.imread(root+'/'+file)

image = cv2.resize(image,(int(img\_w\*1.5), int(img\_h\*1.5)))

image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

Images.append(image)

except Exception as e:

print(e)

Images = np.array(Images)

return (Images, Labels)

def get\_class\_index(Labels):

for i, n in enumerate(Labels):

for j, k in enumerate(CLASSES): # foreach CLASSES

if n == k:

Labels[i] = j

Labels = np.array(Labels)

return Labels

Train\_Imgs, Train\_Lbls = read\_imgs\_lbls(train\_dir)

Train\_Lbls = get\_class\_index(Train\_Lbls)

print('Shape of train images: {}'.format(Train\_Imgs.shape))

print('Shape of train labels: {}'.format(Train\_Lbls.shape))

dim = 4 #you can change it; 4x4 dimension flat plot

f,ax = plt.subplots(dim,dim)

f.subplots\_adjust(0,0,2,2)

for i in range(0,dim):

for j in range(0,dim):

rnd\_number = randint(0,len(Train\_Imgs))

cl = Train\_Lbls[rnd\_number]

ax[i,j].imshow(Train\_Imgs[rnd\_number])

ax[i,j].set\_title(CLASSES[cl]+': ' + str(cl))

ax[i,j].axis('off')

def edge\_and\_cut(img):

try:

edges = cv2.Canny(img, img\_w, img\_h)

if(np.count\_nonzero(edges)>edges.size/10000):

pts = np.argwhere(edges>0)

y1,x1 = pts.min(axis=0)

y2,x2 = pts.max(axis=0)

new\_img = img[y1:y2, x1:x2] # crop the region

new\_img = cv2.resize(new\_img,(img\_w, img\_h)) # Convert back

else:

new\_img = cv2.resize(img,(img\_w, img\_h))

except Exception as e:

print(e)

new\_img = cv2.resize(img,(img\_w, img\_h))

return new\_img

def show\_cropped(img):

emb\_img = img.copy()

edges = cv2.Canny(img, img\_w, img\_h)

if(np.count\_nonzero(edges)>edges.size/10000):

pts = np.argwhere(edges>0)

y1,x1 = pts.min(axis=0)

y2,x2 = pts.max(axis=0)

new\_img = img[y1:y2, x1:x2]

edge\_size = 1 #replace it with bigger size for larger images

emb\_img[y1-edge\_size:y1+edge\_size, x1:x2] = [255, 0, 0]

emb\_img[y2-edge\_size:y2+edge\_size, x1:x2] = [255, 0, 0]

emb\_img[y1:y2, x1-edge\_size:x1+edge\_size] = [255, 0, 0]

emb\_img[y1:y2, x2-edge\_size:x2+edge\_size] = [255, 0, 0]

new\_img = cv2.resize(new\_img,(img\_w, img\_h)) # Convert to primary size

else:

new\_img = cv2.resize(img,(img\_w, img\_h))

fig, ax = plt.subplots(nrows=1, ncols=4, figsize=(10, 10))

ax[0].imshow(img, cmap='gray')

ax[0].set\_title('Original Image', fontsize=14)

ax[1].imshow(edges, cmap='gray')

ax[1].set\_title('Canny Edges', fontsize=14)

ax[2].imshow(emb\_img, cmap='gray')

ax[2].set\_title('Bounding Box', fontsize=14)

ax[3].imshow(new\_img, cmap='gray')

ax[3].set\_title('Cropped', fontsize=14)

for x in range(0,3):

show\_cropped(Train\_Imgs[randint(0,len(Train\_Imgs))])

def crop\_images(Imgs):

CroppedImages = np.ndarray(shape=(len(Imgs), img\_w, img\_h, 3), dtype=np.int)

ind = 0

for im in Imgs:

x = edge\_and\_cut(im)

CroppedImages[ind] = x

ind += 1

return CroppedImages

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

X\_train, X\_val, y\_train, y\_val = train\_test\_split(Train\_Imgs, Train\_Lbls, shuffle = True, test\_size = 0.2, random\_state = 42)

print('Shape of X\_train: {}, y\_train: {} '.format(X\_train.shape, y\_train.shape))

print('Shape of X\_val: {}, y\_val: {} '.format(X\_val.shape, y\_val.shape))  
  
from keras.models import Sequential

from keras.layers import Conv2D, MaxPooling2D, AveragePooling2D

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

from keras import optimizers

EPOCHS =

batch\_size =

iter\_per\_epoch = len(X\_train

val\_per\_epoch = len(X\_val)

model=Sequential()

model.add(Conv2D(filters=96,kernel\_size=(11,11),strides=(4,4),padding="valid",activation="relu",input\_shape=( img\_w, img\_h, 3)))

model.add(MaxPooling2D(pool\_size=(3,3),strides=(2,2)))

model.add(BatchNormalization())

model.add(Conv2D(filters=256,kernel\_size=(5,5),strides=(1,1),padding="valid",activation="relu"))

model.add(MaxPooling2D(pool\_size=(3,3),strides=(2,2)))

model.add(BatchNormalization())

model.add(Conv2D(filters=384,kernel\_size=(3,3),strides=(1,1),padding="valid",activation="relu"))

model.add(Conv2D(filters=384,kernel\_size=(3,3),strides=(1,1),padding="valid",activation="relu"))

model.add(Conv2D(filters=256,kernel\_size=(3,3),strides=(1,1),padding="valid",activation="relu"))

model.add(MaxPooling2D(pool\_size=(3,3),strides=(2,2)))

model.add(BatchNormalization())

model.add(Flatten())

model.add(Dense(4096,input\_shape=(227,227,3),activation="relu"))

model.add(Dropout(0.4))

model.add(BatchNormalization())

model.add(Dense(4096,activation="relu"))

model.add(Dropout(0.4))

model.add(BatchNormalization())

model.add(Dense(1000,activation="relu"))

model.add(Dropout(0.4))

model.add(BatchNormalization())

model.add(Dense(20,activation="softmax"))

model.summary()

model.compile(optimizer='adam',loss='sparse\_categorical\_crossentropy',metrics=['accuracy'])

from keras.preprocessing.image import ImageDataGenerator

train\_datagen = ImageDataGenerator(

rotation\_range=25,

zoom\_range=0.1,

width\_shift\_range=0.1,

height\_shift\_range=0.1,

shear\_range=0.2,

horizontal\_flip=True

)

val\_datagen = ImageDataGenerator

n = randint(0,len(X\_train))

samples = np.expand\_dims(X\_train[n], 0)

it = train\_datagen.flow(samples, batch\_size=batch\_size)

cols = 7

fig, ax = plt.subplots(nrows=1, ncols=cols, figsize=(15, 10))

ax[0].imshow(X\_train[n], cmap='gray')

ax[0].set\_title('Original', fontsize=10)

for i in range(1,cols):

batch = it.next() # generate batch of images

image = batch[0].astype('uint32') # convert to unsigned int for viewing

ax[i].set\_title('augmented {}'.format(i), fontsize=10)

ax[i].imshow(image, cmap='gray')

train\_gen = train\_datagen.flow(X\_train, y\_train, batch\_size=batch\_size)

val\_gen = val\_datagen.flow(X\_val, y\_val, batch\_size=batch\_size)

m = model.fit\_generator(

train\_gen,

steps\_per\_epoch= iter\_per\_epoch,

epochs=EPOCHS,

validation\_data = val\_gen,

validation\_steps = val\_per\_epoch,

verbose = 1

)

fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(15,5))

axs[0].plot(m.history['accuracy'])

axs[0].set\_title('Model accuracy')

axs[0].legend(['Train', 'Val'], loc='upper left')

axs[1].plot(m.history['loss'])

axs[1].set\_title('Model loss')

axs[1].legend(['Train'], loc='upper left')

for ax in axs.flat:

ax.set(xlabel='Epoch')

from sklearn.metrics import confusion\_matrix

y\_pre\_test=model.predict(X\_val)

y\_pre\_test=np.argmax(y\_pre\_test,axis=1)

cm=confusion\_matrix(y\_val,y\_pre\_test)

plt.figure(figsize = (15,15))

sn.heatmap(cm, annot=True)

pred = model.predict(x\_test)

pred.shape

plt.figure(1 , figsize = (19 , 10))

n = 0

for i in range(9):

n += 1

r = np.random.randint( 0, x\_test.shape[0], 1)

plt.subplot(3, 3, n)

plt.subplots\_adjust(hspace = 0.3, wspace = 0.3)

plt.imshow(x\_test[r[0]])

plt.title('Actual = {}, Predicted = {}'.format(y\_test[r[0]] , y\_test[r[0]]\*pred[r[0]][y\_test[r[0]]]) )

plt.xticks([]) , plt.yticks([])

plt.show()

**Executable Code: [Modified CNN]**

import os

import matplotlib.pyplot as plt

import seaborn as sn

import cv2

from random import randint

import numpy as np

CLASSES, bleed = [], [] # names of classes, count of images for each class

for root, dirs, files in os.walk('input/images'):

f = os.path.basename(root) # get class name

if len(files) > 0:

bleed.append(len(files))

if f not in CLASSES:

CLASSES.append(f) # add folder name

ax.legend(fontsize = 12);

bleed\_count = len(CLASSES)

print('{} classes with {} images in total'.format(len(CLASSES), sum(bleed))

img\_w, img\_h = 227, 227 # width and height of image

train\_dir = 'input/images/train/'

def read\_imgs\_lbls(\_dir):

Images, Labels = [], []

for root, dirs, files in os.walk(\_dir):

f = os.path.basename(root) # get class name

for file in files:

Labels.append(f)

try:

image = cv2.imread(root+'/'+file)

image = cv2.resize(image,(int(img\_w\*1.5), int(img\_h\*1.5)))

image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

Images.append(image)

except Exception as e:

print(e)

Images = np.array(Images)

return (Images, Labels)

def get\_class\_index(Labels):

for i, n in enumerate(Labels):

for j, k in enumerate(CLASSES): # foreach CLASSES

if n == k:

Labels[i] = j

Labels = np.array(Labels)

return Labels

Train\_Imgs, Train\_Lbls = read\_imgs\_lbls(train\_dir)

Train\_Lbls = get\_class\_index(Train\_Lbls)

print('Shape of train images: {}'.format(Train\_Imgs.shape))

print('Shape of train labels: {}'.format(Train\_Lbls.shape))

dim = 4 #you can change it; 4x4 dimension flat plot

f,ax = plt.subplots(dim,dim)

f.subplots\_adjust(0,0,2,2)

for i in range(0,dim):

for j in range(0,dim):

rnd\_number = randint(0,len(Train\_Imgs))

cl = Train\_Lbls[rnd\_number]

ax[i,j].imshow(Train\_Imgs[rnd\_number])

ax[i,j].set\_title(CLASSES[cl]+': ' + str(cl))

ax[i,j].axis('off')

def edge\_and\_cut(img):

try:

edges = cv2.Canny(img, img\_w, img\_h)

if(np.count\_nonzero(edges)>edges.size/10000):

pts = np.argwhere(edges>0)

y1,x1 = pts.min(axis=0)

y2,x2 = pts.max(axis=0)

new\_img = img[y1:y2, x1:x2] # crop the region

new\_img = cv2.resize(new\_img,(img\_w, img\_h)) # Convert back

else:

new\_img = cv2.resize(img,(img\_w, img\_h))

except Exception as e:

print(e)

new\_img = cv2.resize(img,(img\_w, img\_h))

return new\_img

def show\_cropped(img):

emb\_img = img.copy()

edges = cv2.Canny(img, img\_w, img\_h)

if(np.count\_nonzero(edges)>edges.size/10000):

pts = np.argwhere(edges>0)

y1,x1 = pts.min(axis=0)

y2,x2 = pts.max(axis=0)

new\_img = img[y1:y2, x1:x2]

edge\_size = 1 #replace it with bigger size for larger images

emb\_img[y1-edge\_size:y1+edge\_size, x1:x2] = [255, 0, 0]

emb\_img[y2-edge\_size:y2+edge\_size, x1:x2] = [255, 0, 0]

emb\_img[y1:y2, x1-edge\_size:x1+edge\_size] = [255, 0, 0]

emb\_img[y1:y2, x2-edge\_size:x2+edge\_size] = [255, 0, 0]

new\_img = cv2.resize(new\_img,(img\_w, img\_h)) # Convert to primary size

else:

new\_img = cv2.resize(img,(img\_w, img\_h))

fig, ax = plt.subplots(nrows=1, ncols=4, figsize=(10, 10))

ax[0].imshow(img, cmap='gray')

ax[0].set\_title('Original Image', fontsize=14)

ax[1].imshow(edges, cmap='gray')

ax[1].set\_title('Canny Edges', fontsize=14)

ax[2].imshow(emb\_img, cmap='gray')

ax[2].set\_title('Bounding Box', fontsize=14)

ax[3].imshow(new\_img, cmap='gray')

ax[3].set\_title('Cropped', fontsize=14)

for x in range(0,3):

show\_cropped(Train\_Imgs[randint(0,len(Train\_Imgs))])

def crop\_images(Imgs):

CroppedImages = np.ndarray(shape=(len(Imgs), img\_w, img\_h, 3), dtype=np.int)

ind = 0

for im in Imgs:

x = edge\_and\_cut(im)

CroppedImages[ind] = x

ind += 1

return CroppedImages

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

X\_train, X\_val, y\_train, y\_val = train\_test\_split(Train\_Imgs, Train\_Lbls, shuffle = True, test\_size = 0.2, random\_state = 42)

print('Shape of X\_train: {}, y\_train: {} '.format(X\_train.shape, y\_train.shape))

print('Shape of X\_val: {}, y\_val: {} '.format(X\_val.shape, y\_val.shape))  
  
from keras.models import Sequential

from keras.layers import Conv2D, MaxPooling2D, AveragePooling2D

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

from keras import optimizers

EPOCHS =

batch\_size =

iter\_per\_epoch = len(X\_train

val\_per\_epoch = len(X\_val)

model = Sequential()

model.add(Conv2D(32, kernel\_size=(3, 3), activation='relu', input\_shape=(227,227,3)))

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

model.add(Dropout(0.25))

model.add(Conv2D(64, kernel\_size=(3, 3), activation='relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Conv2D(64, kernel\_size=(3, 3), activation='relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Conv2D(64, kernel\_size=(3, 3), activation='relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Conv2D(128, kernel\_size=(3, 3), activation='relu'))

model.add(Dropout(0.4))

model.add(Flatten())

model.add(Dense(128, activation='relu'))

model.add(Dropout(0.3))

model.add(Dense(10, activation='softmax'))

model.summary()

model.compile(optimizer='adam',loss='sparse\_categorical\_crossentropy',metrics=['accuracy'])

from keras.preprocessing.image import ImageDataGenerator

train\_datagen = ImageDataGenerator(

rotation\_range=25,

zoom\_range=0.1,

width\_shift\_range=0.1,

height\_shift\_range=0.1,

shear\_range=0.2,

horizontal\_flip=True

)

val\_datagen = ImageDataGenerator

n = randint(0,len(X\_train))

samples = np.expand\_dims(X\_train[n], 0)

it = train\_datagen.flow(samples, batch\_size=batch\_size)

cols = 7

fig, ax = plt.subplots(nrows=1, ncols=cols, figsize=(15, 10))

ax[0].imshow(X\_train[n], cmap='gray')

ax[0].set\_title('Original', fontsize=10)

for i in range(1,cols):

batch = it.next() # generate batch of images

image = batch[0].astype('uint32') # convert to unsigned int for viewing

ax[i].set\_title('augmented {}'.format(i), fontsize=10)

ax[i].imshow(image, cmap='gray')

train\_gen = train\_datagen.flow(X\_train, y\_train, batch\_size=batch\_size)

val\_gen = val\_datagen.flow(X\_val, y\_val, batch\_size=batch\_size)

m = model.fit\_generator(

train\_gen,

steps\_per\_epoch= iter\_per\_epoch,

epochs=EPOCHS,

validation\_data = val\_gen,

validation\_steps = val\_per\_epoch,

verbose = 1

)

fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(15,5))

axs[0].plot(m.history['accuracy'])

axs[0].set\_title('Model accuracy')

axs[0].legend(['Train', 'Val'], loc='upper left')

axs[1].plot(m.history['loss'])

axs[1].set\_title('Model loss')

axs[1].legend(['Train'], loc='upper left')

for ax in axs.flat:

ax.set(xlabel='Epoch')

from sklearn.metrics import confusion\_matrix

y\_pre\_test=model.predict(X\_val)

y\_pre\_test=np.argmax(y\_pre\_test,axis=1)

cm=confusion\_matrix(y\_val,y\_pre\_test)

plt.figure(figsize = (15,15))

sn.heatmap(cm, annot=True)

pred = model.predict(x\_test)

pred.shape

plt.figure(1 , figsize = (19 , 10))

n = 0

for i in range(9):

n += 1

r = np.random.randint( 0, x\_test.shape[0], 1)

plt.subplot(3, 3, n)

plt.subplots\_adjust(hspace = 0.3, wspace = 0.3)

plt.imshow(x\_test[r[0]])

plt.title('Actual = {}, Predicted = {}'.format(y\_test[r[0]] , y\_test[r[0]]\*pred[r[0]][y\_test[r[0]]]) )

plt.xticks([]) , plt.yticks([])

plt.show()