APPENDIX

CODING

GABOR FILTER FUNCTION

```
1. def gaborFilter(img):
2.     filter = []
3.     num_filters = 16
4.     ksize = 35
5.     sigma = 10.0
6.     lmbda = 30.0
7.     gamma = 0.5
8.     psi = 0
9.     for theta in np.arange(0, np.pi, np.pi / num_filters):
10.      krn = cv2.getGaborKernel((ksize, ksize), sigma, theta, lmbda,
         gamma, psi, ktype=cv2.CV_64F)
11.      krn /= 1.0 * krn.sum()
12.      filter.append(krn)
13.    filteredImage = np.zeros_like(img)
14.    depth = -1
15.    for krn in filter:
16.      filtering = cv2.filter2D(img, depth , krn)
17.      np.maximum(filteredImage, filtering, filteredImage)
18.    return filteredImage
```

CHAN-VESE SEGMENTATION FUNCTION

```
1. def chanveseFilter(img):
2.     cv = chan_vese(img, mu=0.25, lambda1=1, lambda2=1, tol=1e-3,
       max_num_iter=200, dt=0.5,
       init_level_set="checkerboard",extended_output=True)
3.     return cv
```

COMBINED FEATURE / BLEND IMAGE FUNCTION

```
1. def blendImage(imgGabor,imgCV):
2.     imgCV = 255 - imgCV
3.     blendImg = cv2.subtract(imgCV, imgGabor, dtype=cv2.CV_8U)
4.     return blendImg
```

IMAGE PREPROCESSING FUNCTION

```
1. def imgPreProcess(img):
2.     gbImg = gaborFilter(img)
3.     cvImg = chanveseFilter(img)
4.     resultImg = blendImage(np.asarray(cvImg[1]),gbImg)
5.     resultImg = 2.2 * resultImg + 75
6.     return resultImg
```

ONE-HOT ENCODING CONVERSION FUNCTION

```
1. def convert_to_one_hot(Y, C):
2.     Y = np.eye(C)[Y.reshape(-1)].T
3.     return Y
```

DATA LOAD FUNCTION

```
1. def loadData(DATA_DIR):
2.     DATA_Images = []
3.     progress = 1
4.     for imgFile in os.listdir(DATA_DIR):
5.         print('Image File Name : ',imgFile)
6.         tempHolder1 = cv2.imread(os.path.join(DATA_DIR,imgFile))
7.         tempHolder1 = cv2.cvtColor(tempHolder1, cv2.COLOR_BGR2YCR_CB)
8.           tempHolder2          =          cv2.resize(tempHolder1[:,:,1
       (200,200),interpolation=cv2.INTER_AREA)
9.         print("  Success Reading Image ...")
10.        imgProcess = imgPreProcess(tempHolder2)
11.        print("  Success Processing Image ...(",progress,")")
12.        progress += 1
13.        if 'au' in imgFile.lower():
14.            DATA_Images.append([imgProcess,1])
15.        elif 'tp' in imgFile.lower():
16.            DATA_Images.append([imgProcess,0])
17.    TRAIN_img, TEST_img = train_test_split(DATA_Images, test_size=0.2,
       random_state=25)
18.    train_set_x = []
19.    train_set_y = []
20.    test_set_x = []
21.    test_set_y = []
22.    for xTr,yTr in TRAIN_img:
23.        train_set_x.append(xTr)
24.        train_set_y.append(yTr)
25.    for xTe,yTe in TEST_img:
26.        test_set_x.append(xTe)
27.        test_set_y.append(yTe)
28.    return train_set_x, train_set_y, test_set_x, test_set_y
```

LOAD AND PREPARE DATASET

```
1. ROWS = 200
2. COLS = 200
3. CHANNELS = 1
4. CLASSES = 2
5.
6. DATA_DIRECT = "C:\\Users\\...\\ProjectDocument\\TrainValData\\"
7. train_set_x, train_set_y, test_set_x, test_set_y = loadData(DATA_DIRECT)
8. train_set_x = np.array(train_set_x,dtype="uint8")
9. train_set_y = np.array(train_set_y,dtype="uint8")
10.test_set_x = np.array(test_set_x,dtype="uint8")
11.test_set_y = np.array(test_set_y,dtype="uint8")
12.X_train = train_set_x/255
13.X_test = test_set_x/255
```

```
14.Y_train = convert_to_one_hot(train_set_y, CLASSES).T
15.Y_test = convert_to_one_hot(test_set_y, CLASSES).T
```

IDENTITY BLOCK FUNCTION

```
1. def identity_block(X, f, filters, stage, block):
2.     conv_name_base = 'res' + str(stage) + block + '_branch'
3.     bn_name_base = 'bn' + str(stage) + block + '_branch'
4.
5.     F1, F2, F3 = filters
6.
7.     X_shortcut = X
8.
9.     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)
10.    X = BatchNormalization(axis = 3, name = bn_name_base + '2a')(X)
11.    X = Activation('relu')(X)
12.
13.    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)
14.    X = BatchNormalization(axis = 3, name = bn_name_base + '2b')(X)
15.    X = Activation('relu')(X)

16.    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)
17.    X = BatchNormalization(axis = 3, name = bn_name_base + '2c')(X)
18.
19.    X = Add()([X, X_shortcut])
20.    X = Activation('relu')(X)
21.
22.    return X
```

CONVOLUTIONAL BLOCK FUNCTION

```
1. def convolutional_block(X, f, filters, stage, block, s = 2):
2.     conv_name_base = 'res' + str(stage) + block + '_branch'
3.     bn_name_base = 'bn' + str(stage) + block + '_branch'
4.
5.     F1, F2, F3 = filters
6.
7.     X_shortcut = X
8.
9.
10.    X = Conv2D(F1, (1, 1), strides = (s,s), name = conv_name_base +
      '2a', kernel_initializer = glorot_uniform(seed=0))(X)
11.    X = BatchNormalization(axis = 3, name = bn_name_base + '2a')(X)
12.    X = Activation('relu')(X)
13.
14.    X = Conv2D(filters=F2, kernel_size=(f, f), strides=(1, 1),
       padding='same',name=conv_name_base+'2b',kernel_initializer=glorot_un
       iform(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_u
    niform(seed=0))(X)
    X = BatchNormalization(axis=3, name=bn_name_base + '2c')(X)

    X_shortcut = Conv2D(F3, (1, 1), strides = (s,s), 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
```

RESNET50 FUNCTION

```
def ResNet50(input_shape = (200, 200, 1), classes = 2):
    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 = 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((2, 2), name='avg_pool')(X)

    X = Flatten()(X)
    X = Dense(classes, activation='softmax', name='fc' + str(classes),
    kernel_initializer = glorot_uniform(seed=0))(X)

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

    return model
```

BUILD RESNET50 MODEL

```
model = ResNet50(input_shape = (ROWS, COLS, CHANNELS), classes =
CLASSES)

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

K.set_value(model.optimizer.learning_rate, 0.0000001)

model.fit(X_train, Y_train, epochs = 50, batch_size = 16)

preds = model.evaluate(X_test, Y_test)
print ("Loss = " + str(preds[0]))
print ("Test Accuracy = " + str(preds[1]))

model.summary()

model.save('ResNet50.h5')
```

Similarity Report

PAPER NAME

TA-18.K1.0016.docx

WORD COUNT	CHARACTER COUNT
4270 Words	**19917 Characters**
PAGE COUNT	FILE SIZE
7 Pages	**20.0KB**
SUBMISSION DATE	REPORT DATE
Dec 19, 2022 11:07 AM GMT+7	**Dec 19, 2022 11:08 AM GMT+7**

1% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

- 0% Internet database
- Crossref database
- 0% Submitted Works database
- 0% Publications database
- Crossref Posted Content database

Summary