Importing Libraries

The 2D Global average pooling block takes a tensor of size (input width) x (input height) x (input channels) and computes the average value of all values across the entire (input width) x (input height) matrix for each of the (input channels).
Use global average pooling blocks as an alternative to the Flattening block after the last pooling block of your convolutional neural network. Using 2D Global average pooling block can replace the fully connected blocks of your CNN.

```
import numpy as np
import os
import time
from keras.preprocessing import image
from keras.layers import GlobalAveragePooling2D, Dense, Dropout,Activation,Flatten
import tensorflow as tf

from keras.applications.vgg16 import preprocess_input
from keras.layers import Input
from keras.models import Model
from keras.utils import np_utils
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split

from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import confusion_matrix



from google.colab import drive
drive.mount('/content/drive')
```

1. Keras. layers. flatten function flattens the multi-dimensional input tensors into a single dimension, so you can model your input layer and build your neural network model, then pass those data into every single neuron of the model

2. Keras works with batches of images. So, the first dimension is used for the number of samples (or images) you have.

When you load a single image, you get the shape of one image, which is (size1,size2,channels).

In order to create a batch of images, you need an additional dimension: (samples, size1,size2,channels)

The preprocess_input function is meant to adequate your image to the format the model requires.

Some models use images with values ranging from 0 to 1. Others from -1 to +1. Others use the "caffe" style, that is not normalized, but is centered.

3. np_utils. to_categorical is used to convert array of labeled data(from 0 to nb_classes - 1) to one-hot vector.

4. Python Imaging Library is a free and open-source additional library for the Python programming language that adds support for opening, manipulating, and saving many different image file formats.

Loading and preprocessing Data

```
# Loading the training data
PATH = '/content/drive/MyDrive/Resnet6'
# Define data path
data_path = PATH + '/data'
data_dir_list = ['Crack','NoCrack']     Mask,No mask
img_data_list=[]

img_data_list=[]

for dataset in data_dir_list:
    img_list=os.listdir(data_path+"/"+ dataset)
    #print ('Loaded the images of dataset-'+'{}\n'.format(dataset))
    for img in img_list:
        img_path = data_path + '/'+ dataset + '/'+ img
        imge = image.load_img(img_path, target_size=(224, 224))
        xab = image.img_to_array(imge)
        xab = preprocess_input(xab)
        img_data_list.append(xab)
```

Either None (default to original size) or tuple of ints (img_height, img_width).

Converts a PIL Image instance to a Numpy array.

```
img_data = np.array(img_data_list)
#img_data = img_data.astype('float32')
print (img_data.shape)
#img_data=np.rollaxis(img_data,1,0)
print (img_data.shape)
#img_data=img_data[0]
print (img_data.shape)
```

Splitting Dataset

```
# Define the number of classes
num_classes = 2
num_of_samples = img_data.shape[0]
labels = np.ones((num_of_samples,),dtype='int64')

labels[0:int(num_of_samples/2)]=0
labels[int(num_of_samples/2):num_of_samples]=1
```

We have set up samples in such a way that first half of images contains 1 person and next half contains 2 person

```
names = ['Crack','No Crack']
```

['Mask','No Mask']

```
# convert class labels to on-hot encoding
Y = np_utils.to_categorical(labels, num_classes)
```

Using the method to_categorical(), a numpy array (or) a vector which has integers that represent different categories, can be converted into a numpy array (or) a matrix which has binary values and has columns equal to the number of categories in the data.

```
#Shuffle the dataset
x,y = shuffle(img_data,Y, random_state=2)
# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.15, random_state=2)
```

sklearn.utils.shuffle(*arrays, random_state=None, n_samples=None)
n_samples: Number of samples to generate. If left to None this is automatically set to the first dimension of the arrays. It should not be larger than the length of arrays.

Training the resnet model

```
#Training the classifier alone
image_input = Input(shape=(224,224, 3))

model = tf.keras.applications.ResNet50(input_tensor=image_input)
#model.summary()
last_layer = model.get_layer('avg_pool').output
x= Flatten(name='flatten')(last_layer)
out = Dense(num_classes, activation='softmax', name='output_layer')(x)
custom_resnet_model = Model(inputs=image_input,outputs= out)
#custom_resnet_model.summary()

for layer in custom_resnet_model.layers[:-1]:
```

```
    layer.trainable = False

custom_resnet_model.layers[-1].trainable

custom_resnet_model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accura
Noepochs=20
t=time.time()
hist = custom_resnet_model.fit(X_train, y_train, batch_size=32, epochs=Noepochs, verbose=1, v
print('Training time: %s' % (t - time.time()))
(loss, accuracy) = custom_resnet_model.evaluate(X_test, y_test, batch_size=10, verbose=1)

print("[INFO] loss={:.4f}, accuracy: {:.4f}%".format(loss,accuracy * 100))
```

Predicting and dividing images into different folders

```
import shutil
predict_images = os.listdir('/content/drive/MyDrive/Resnet6/predict/data')
pred_data_list = []
print(predict_images)
for img in predict_images:
        imge = image.load_img('/content/drive/MyDrive/Resnet6/predict/data/'+ img, target_siz
        x = image.img_to_array(imge)
        x = preprocess_input(x)
        pred_data_list.append(x)

pred_data = np.array(pred_data_list)
print(img_data.shape)
data_class = custom_resnet_model.predict(pred_data)
i=0
while(i<len(data_class)):
    if(data_class[i][0]<=0.5):
          data_class[i][0]=0
    else:
          data_class[i][0]=1
    if(data_class[i][1]<=0.5):
          data_class[i][1]=0
    else:
          data_class[i][1]=1

    i+=1

shutil.rmtree('/content/drive/MyDrive/Resnet6/predict/Cracks/',ignore_errors=True)
shutil.rmtree('/content/drive/MyDrive/Resnet6/predict/NoCracks/',ignore_errors=True)
os.makedirs('/content/drive/MyDrive/Resnet6/predict/Cracks/')
os.makedirs('/content/drive/MyDrive/Resnet6/predict/NoCracks/')
```

```
i=0
for j in data_class:
    if j[0]==1:
        shutil.copyfile('/content/drive/MyDrive/Resnet6/predict/data/'+predict_images[i], '/c
    elif j[1]==1:
        shutil.copyfile('/content/drive/MyDrive/Resnet6/predict/data/'+predict_images[i], '/c
    i+=1
```

Visualizing results

```
aaa = custom_resnet_model.predict(X_test)
i=0
while(i<len(aaa)):
    if(aaa[i][0]<=0.5):
          aaa[i][0]=0
    else:
          aaa[i][0]=1
    if(aaa[i][1]<=0.5):
          aaa[i][1]=0
    else:
          aaa[i][1]=1
    i+=1
```

```
import matplotlib.pyplot as plt
# visualizing losses and accuracy
train_loss=hist.history['loss']
val_loss=hist.history['val_loss']
train_acc=hist.history['accuracy']
val_acc=hist.history['val_accuracy']
xc=range(Noepochs)

plt.figure(1,figsize=(8,5))
plt.plot(xc,train_loss)
plt.plot(xc,val_loss)
plt.xlabel('num of Epochs')
plt.ylabel('loss')
plt.title('train_loss vs val_loss')
plt.grid(True)
plt.legend(['train','val'])
#print plt.style.available # use bmh, classic,ggplot for big pictures
plt.style.use(['classic'])

plt.figure(2,figsize=(8,5))
plt.plot(xc,train_acc)
plt.plot(xc,val_acc)
```

```
plt.xlabel('num of Epochs')
plt.ylabel('accuracy')
plt.title('train_acc vs val_acc')
plt.grid(True)
plt.legend(['train','val'],loc=4)
#print plt.style.available # use bmh, classic,ggplot for big pictures
plt.style.use(['classic'])
plt.show(block=False)
```

```
cm = confusion_matrix(y_test.argmax(axis=1), aaa.argmax(axis=1))
ac = y_test.argmax(axis=1)
pc = aaa.argmax(axis=1)
tp =0
for i,j in zip(ac,pc):
    if(i==j==0):
        tp+=1
tn =0
for i,j in zip(ac,pc):
    if(i==j==1):
        tn+=1
fn =0;
for i,j in zip(ac,pc):
    if(i!=j==1):
        fn+=1
fp =0;
for i,j in zip(ac,pc):
    if(i!=j==0):
        fp+=1
pres = tp/(tp+fp)
recall = tp/(tp+fn)
fmeasure = (2*pres*recall)/(pres+recall)

disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=['Crack','NoCrack'])

disp.plot(cmap=plt.cm.Blues)
plt.show()
print("Precision: ","{:.3f}".format(pres))
print("Recall: ","{:.3f}".format(recall))
print("F-measure","{:.3f}".format(fmeasure))
```

YOLOv5 : It is a novel convolutional neural network (CNN) that detects objects in real-time with great accuracy. This approach uses a single neural network to process the entire picture, then separates it into parts and predicts bounding boxes and probabilities for each component. These bounding boxes are weighted by the expected probability. The method “just looks once” at the image in the sense that it makes predictions after only one forward propagation run through the neural network. It then delivers detected items after non-max suppression (which ensures that the object detection algorithm only identifies each object once).

Model Backbone is mostly used to extract key features from an input image. CSP(Cross Stage Partial Networks) are used as a backbone in YOLO v5 to extract rich in useful characteristics from an input image.

The Model Neck is mostly used to create feature pyramids. Feature pyramids aid models in generalizing successfully when it comes to object scaling. It aids in the identification of the same object in various sizes and scales

try to make sure that the number of objects in each class is evenly distributed..