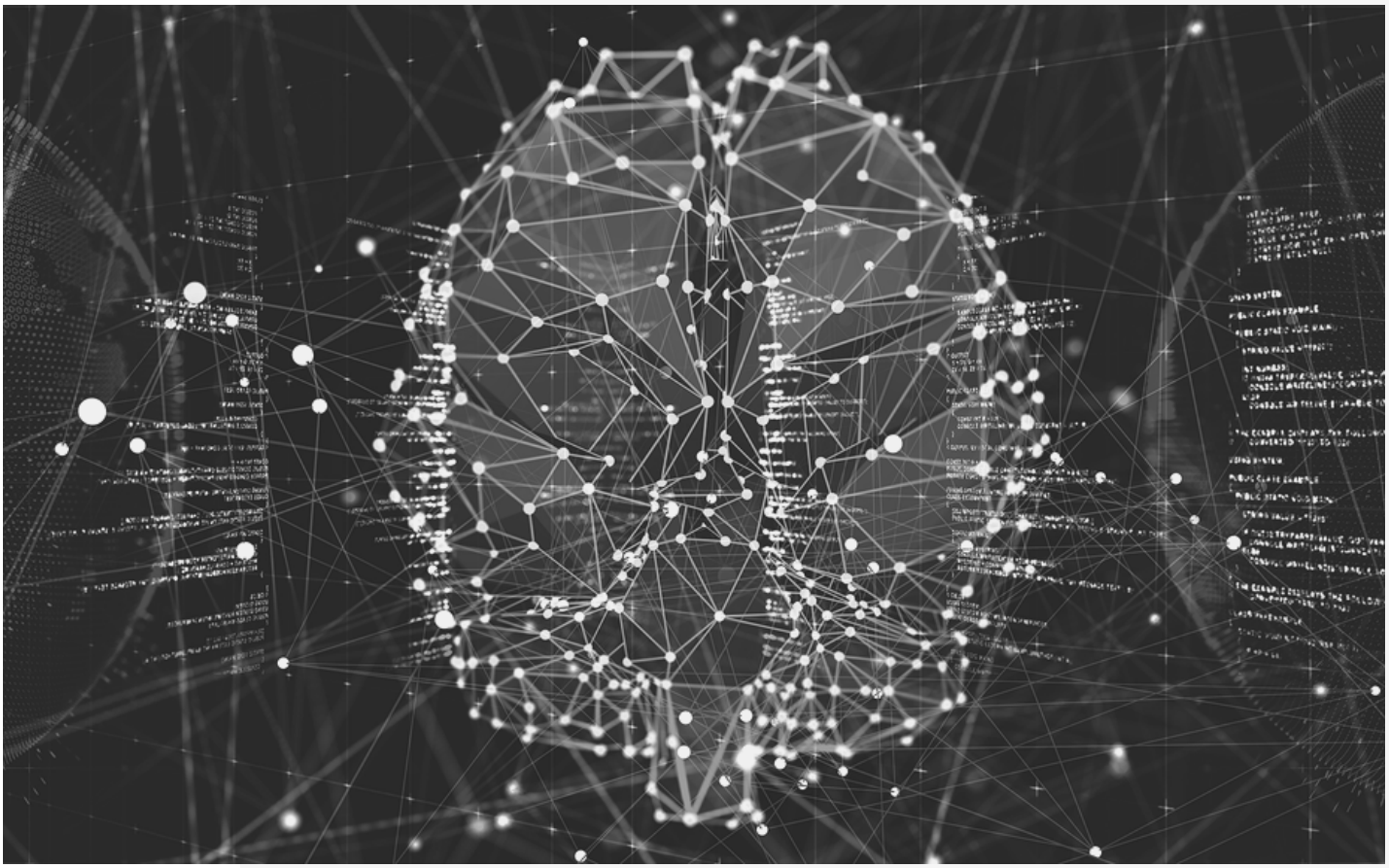
CE 6051: MACHINE LEARNING IN CIVIL ENGINEERING

PROJECT REPORT

IMAGE CLASSIFICATION IN CONSTRUCTION



GROUP 12

	Contribution
CE21M018 : REEMA RAZZAC	17%
CE21M022 : VINEETH R	32%
CE21M114 : HRITIK R V	17%
CE21M117 : KUMARESAN P	17%
CE21M123 : S LALIT KUMAR	17%

CONTENTS

INTRODUCTION

MACHINE LEARNING:

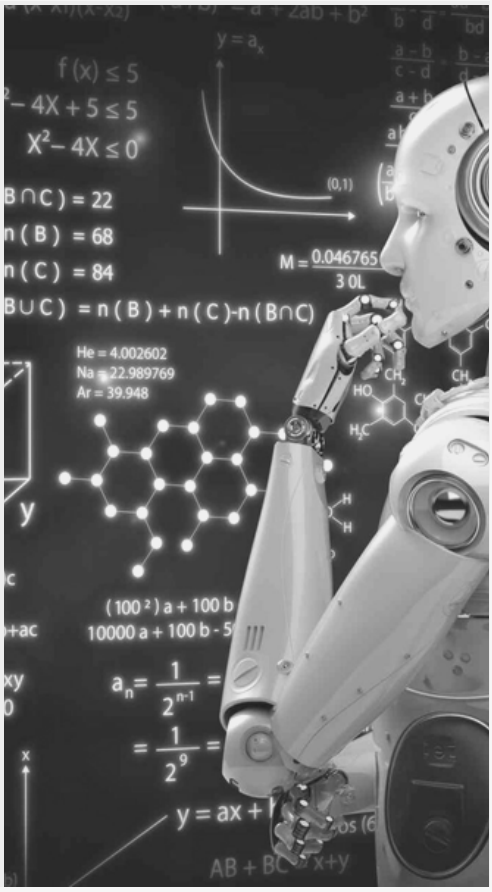
Machine learning (ML), as the name suggests, gives machines the ability to learn. To put it in a formal way, machine learning is the field of study that gives computers the capability to learn from data without being explicitly programmed. It is a subfield of AI that evolved from pattern recognition and computational learning theory. ML already plays a major role in our day-to-day lives. From product recommendations, youtube recommendations, to stock market predictions, speech recognition, and image classification, ML acts as the backbone of all these applications.

IMAGE CLASSIFICATION:

In this wide range of applications, image recognition and classification is a complex problem to solve. Image recognition is the task of identifying what a given image represents. It is a supervised learning problem i.e., the output is a class label. An image classification model can be trained to recognize various class labels. For instance, image classification can be trained to recognize vehicles like cars, trucks, and motorbikes. Image classification is applied in various fields of medical imaging, traffic control systems, object identification from satellite images, and so on.

CONSTRUCTION INDUSTRY AND SUSTAINABILITY

Sustainability has become a significant concern for the past several years and has affected the functioning of many industries. The construction industry is no exception to this as the industry's functioning is unsustainable in many ways. Research points out that the construction industry consumes around 30 to 50% of all extracted natural resources and contributes to 25% of solid waste generation throughout the world. The industry is responsible for around 23% of the total carbon emissions. All these factors make the construction industry one of the significant contributors to environmental degradation and measures to reduce this impact is the need of the hour.



SUSTAINABLE PRACTICES

Though it is not possible to shift the working of the industry to adopt all the sustainable practices, following a few simple measures at the site could contribute substantially towards sustainable construction practices. One such measure is the recycling of materials used at the site for construction. Construction uses lots of materials like cement, sand, steel reinforcement, bricks, and so on. A major portion of these materials is wasted in one way or another and is left as debris around the site. Research has shown that the debris quantity could be as high as 10 to 15% of the materials used in the construction process. On the other hand, there is a company that saved 96% of waste from landfills by using sophisticated recycling and reuse methods. Therefore, it is clear that there is a huge potential for material recycling in the construction process.

PROBLEM STATEMENT

Identifying recyclable materials within construction debris is still a problem and most recyclable materials could be left unnoticed. Therefore, in this project, the problem statement is to identify recyclable objects such as bricks and rebars within construction debris through image processing.

OBJECTIVES OF THIS PROJECT

1. To build and train an image classification ML model that will predict brick and rebar present in a given image

2. To improve the accuracy of the model by feeding augmented images as the training dataset

3. To develop and compare the accuracy results of different image classification algorithms

LITERATURE REVIEW

Review of deep learning: concepts, CNN architectures, challenges, applications, future directions - This review paper attempts to provide a more comprehensive survey of the most important aspects of deep learning, specifically convolutional neural networks. It describes the development of CNNs architectures together with their main features and concludes with challenges and existing research gaps in this line of research.

The classification of construction waste material using a deep convolutional neural network - The paper identifies on-site waste management as a complex problem and proposes on-site waste sorting using technologies that automatically identify different materials. This research aimed to design and describe a deep convolutional neural network (CNN) to identify 7 typical construction & demolition waste classifications (single and mixed disposal) using digital images of waste deposited in a construction site bin. The experiments delivered 94% accuracy, classifying both single and mixed construction & demolition waste.

Waste image classification based on transfer learning and convolutional neural network - The paper explains the importance of having an intelligent waste classification system and identifies that the low accuracy of traditional waste classification is a problem. To improve the efficiency the paper proposes a DenseNet169 waste image classification model using transfer learning and CNN. Classification accuracy of over 82% is achieved in the DenseNet169 model after the transfer learning.

A review of YOLO algorithm developments - This research paper gives a brief overview of the You Only Look Once (YOLO) algorithm and its subsequent advanced versions. This paper briefly describes the development process of the YOLO algorithm, summarizes the methods of target recognition and feature selection, and provides literature support for the targeted picture news and feature extraction in the financial and other fields. The results show the differences and similarities between the YOLO versions and between YOLO and CNN algorithms. The main insight is that the YOLO algorithm improvement is still ongoing.

Transfer learning for multi-crop leaf disease image classification using convolutional neural network VGG - This paper identifies diseased leaves identification and classification especially in tomato and grape plants as a problem in agriculture. To tackle this problem the paper proposes the use of CNN methods for detecting multi-crop leaf disease. It also uses CNN-based Visual Geometry Group (VGG) for improved performance measures. Classification accuracy of 98.4% of grapes and 95.71% of tomatoes is achieved.

EXECUTIVE SUMMARY

The construction industry is the slowest among the sectors in adapting to new-age technology, with sustainability at the forefront amalgamation of technology and conventional practices necessary to stay relevant. Research points out that the construction industry consumes around 30 to 50% of all extracted natural resources and contributes to 25% of solid waste generation throughout the world and it is also responsible for around 23% of the total carbon emissions. Private and governmental organizations find it difficult to manage a huge fraction of construction debris generated every day, mostly it is dumped in landfill or left unnoticed leading to the loss of useful materials. The construction industry can achieve a closed-loop system only when they realize the potential of construction debris. The objective of this project was to develop a model to classify construction debris into Brick and Rebar.

Understanding the data provided is a crucial step in creating good models, sizes of the images provided were identified, and it is necessary to have the same size for brick and rebar so a relevant size was selected to result in the neglection of fewer images. CNN was the first model used for the classification, even after having higher accuracy in the training set, it had a lot of misclassifications in the test dataset. Dataset was increased by performing image augmentation, augmented images combined with annotations were provided to YOLO V5, it could identify the object of concern but was not able to put bounding boxes over every brick or rebar in the image, but it conveyed the information that indeed the debris contains the object of interest. Random Forest is one of the most powerful classification algorithms, augmented images were provided for the classification but it had lower accuracy in training and test dataset. VGG19 was the last model that was used for the classification problem, it has a very deep combination of convolution and max pooling layers, which enables the identification of brick and rebar features. It had very high training and test accuracy hence it was chosen as the final model for prediction. Models like VGG19 help construction practitioners identify useful materials from construction debris in a remote way and enable the transition to a circular built environment.

BRICK & REBAR CLASSIFICATION

1. **Data Preparation**
   Image Statistics and Resizing brick and rebar images
2. **CNN**
   Without Image augmentation, high training accuracy but had lot of misclassification in test data
3. **Yolo V5**
   Can identify brick and rebar, but can't put bounding box over all the objects of concern
4. **Random Forest**
   With Image augmentation, less accuracy in train and test dataset
5. **VGG19**
   With Image augmentation, very high training and test accuracy, chosen as the final model

SIZE STATISTICS

Preparing a dataset is a key aspect of any ML problem, with the brick and rebar images it was necessary to identify the sizes of various images provided and resize them in a way that does not lose crucial information. Steps involved in identifying the images sizes

CODE SNIPPETS

Importing necessary libraries

```
from PIL import Image
import os,sys
from os.path import exists
from keras.preprocessing.image import ImageDataGenerator
import scipy.ndimage
from scipy import ndimage
```

Code to read the image sizes of Rebar

```
width_rebar=[0]*125
height_rebar=[0]*125
sizes=[]*125
path = '/content/drive/MyDrive/rebars/'
print(path)
files = os.listdir(path)
i=0
files=files[0:-1]
for filename in files:
  img=Image.open('/content/drive/MyDrive/rebars/'+filename)
  w,h=img.size
  width_rebar[i]=w
  height_rebar[i]=h
  sizes.append([w,h])
  i=i+1
```

Information of No. of Rebar images greater than a certain size

```
count=0
ind=[]
file_needed=[]
for j in range(len(ranges)):
  for i in range(len(sizes)):
    if(sizes[i][0]>=ranges[j] and sizes[i][1]>=ranges[j]):
      count=count+1
      file_needed.append(files[i])
  print("no.of rebar images having size greater than "+str(ranges[j])+" X "+str(ranges[j])+" is - ",count)
  count=0
```

```
no.of rebar images having size greater than 100 X 100 is -  124
no.of rebar images having size greater than 150 X 150 is -  124
no.of rebar images having size greater than 200 X 200 is -  122
no.of rebar images having size greater than 250 X 250 is -  120
no.of rebar images having size greater than 300 X 300 is -  115
no.of rebar images having size greater than 350 X 350 is -  113
no.of rebar images having size greater than 400 X 400 is -  109
no.of rebar images having size greater than 450 X 450 is -  107
no.of rebar images having size greater than 500 X 500 is -  104
no.of rebar images having size greater than 550 X 550 is -  104
no.of rebar images having size greater than 600 X 600 is -  99
```

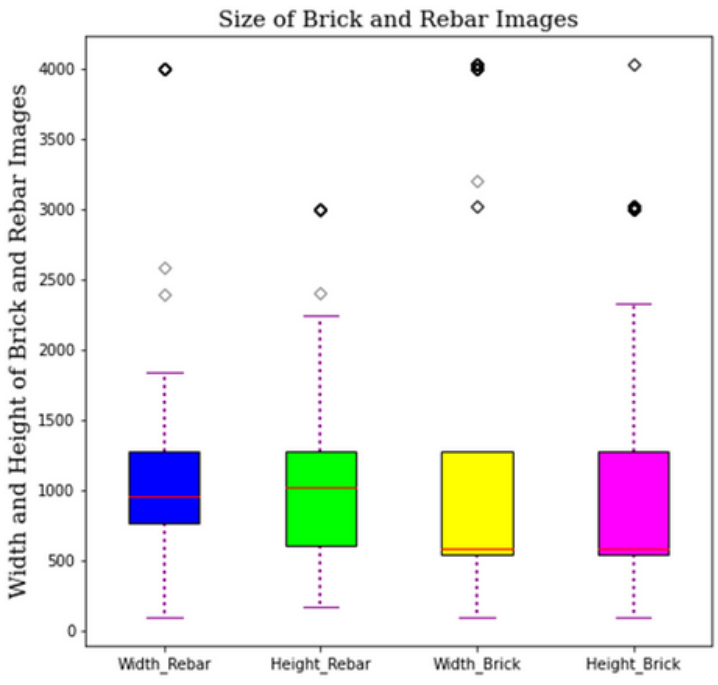
Information of No. of Rebar images greater than a certain size

```
count=0
ind=[]
file_needed_bricks=[]
for j in range(len(ranges)):
  for i in range(len(sizes_brick)):
    if(sizes_brick[i][0]>=ranges[j] and sizes_brick[i][1]>=ranges[j]):
      count=count+1
      file_needed_bricks.append(files[i])
  print("no.of brick images having size greater than "+str(ranges[j])+" X "+str(ranges[j])+" is - ",count)
  count=0
```

```
no.of brick images having size greater than 100 X 100 is -  211
no.of brick images having size greater than 150 X 150 is -  161
no.of brick images having size greater than 200 X 200 is -  161
no.of brick images having size greater than 250 X 250 is -  161
no.of brick images having size greater than 300 X 300 is -  161
no.of brick images having size greater than 350 X 350 is -  161
no.of brick images having size greater than 400 X 400 is -  161
no.of brick images having size greater than 450 X 450 is -  161
no.of brick images having size greater than 500 X 500 is -  161
no.of brick images having size greater than 550 X 550 is -  158
no.of brick images having size greater than 600 X 600 is -  95
```

A similar code was used to identify the image sizes of Brick. The idea was if the smallest image size does not differ much from the largest image size, then we could resize it to the smallest, else we must find one common size for both brick and rebar which would result in the neglection of fewer images.

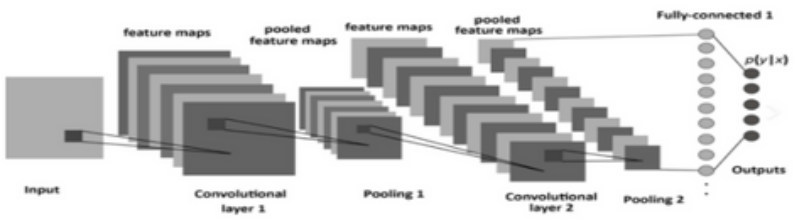
From the analysis, we found a large variation in the size of the images provided, which is evident from the box plot, 50th percentile of rebar images is smaller than brick images, we decided to resize all the images to 500 x 500, which would result in throwing away fewer images.



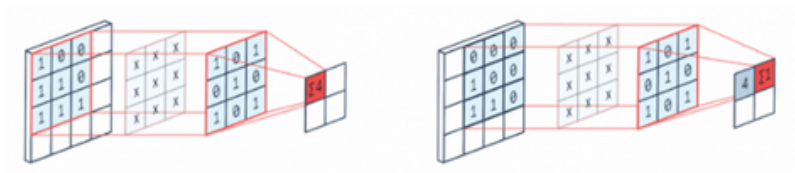
CONVOLUTIONAL NEURAL NETWORK

Convolutional neural networks are particularly successful in finding patterns in an input image because they can readily pickup lines, gradients, circles, etc. This approach may be used to raw photos without any special preprocessing, making it hassle-free. A feed-forward neural network with convolutional layers that can individually recognize complex aspects of the image piled on top of one another is used here. We employed a CNN method with two convolutional and max pooling layers, two dense layers, and one layer.

BASIC CNN ARCHITECTURE



A CNN architecture is formed by stacking convolutional layers, pooling layers and fully connected layers.



In the figure above a 3X3 convolutional kernel is used on a 4X4 image giving a 2X2 convoluted image. The convoluted output will contain data about the image features and is called the feature map. This feature map can be passed on to other layers to learn other characteristics of the image.
The features' spatial size is diminished by the pooling layer, which is placed between the convolutional layers. The largest value among all the values of a certain portion of the matrix is taken in max pooling.
The CNN's final layer is the fully connected layer. It links the neurons between several layers and consists of biases and weights.
One of the CNN model's most crucial components, activation functions are utilized to approximate complex relationships between variables. ReLU, Softmax, tanH, and sigmoid functions are frequently used as activation functions in the CNN algorithm.

WHY USE CNN?

The primary advantage of using CNN is that it can detect and learn important features on its own without external help, which is necessary for the problem statement discussed here. CNN is computationally efficient as it can perform convolutions pooling and parameter sharing and also it can run on any devise making it universally attractive.

CODE SNIPPETS

Extracting resized images of bricks and rebar

```
[10] fil2=[]
     path = '/content/drive/MyDrive/bricks/bricks_resized'
     print(path)
     files = os.listdir(path)
     for filename in files:
         fil2.append(filename)
     print(fil2)

[6]  fil1=[]
     path = '/content/drive/MyDrive/rebars/rebar_resized'
     print(path)
     files = os.listdir(path)
     for filename in files:
         fil1.append(filename)
     print(fil1)
```

Annotating and converting the images to numpy array

```
[17] X=[]
     y = list()
     for i in range(len(fil3)):
       img_path ='/content/drive/MyDrive/for_cnn/'+fil3[i]
       if(fil3[i][0:6]=='bricks'):
         correct_cat = 0
       else:
         correct_cat=1
       img = load_img('/content/drive/MyDrive/for_cnn/'+fil3[i])
       data = img_to_array(img)
       X.append(data)
       y.append(correct_cat)
```

Train, Test and split data: In the code snippet below stratify is given so that when the images are split, it is always split in the same way

```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.20, random_state=1, stratify=y)
```

Structure of Convolution followed by dense layer

```
x_train=x_train/255
x_test=x_test/255
cnn=models.Sequential([
    layers.Conv2D(filters=32,kernel_size=(3,3),activation='relu',input_shape=(500,500,3)),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(filters=16,kernel_size=(3,3),activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Flatten(),
    layers.Dense(64,activation='relu'),
    layers.Dense(2,activation='softmax')
])
```

CNN training and accuracy across epochs

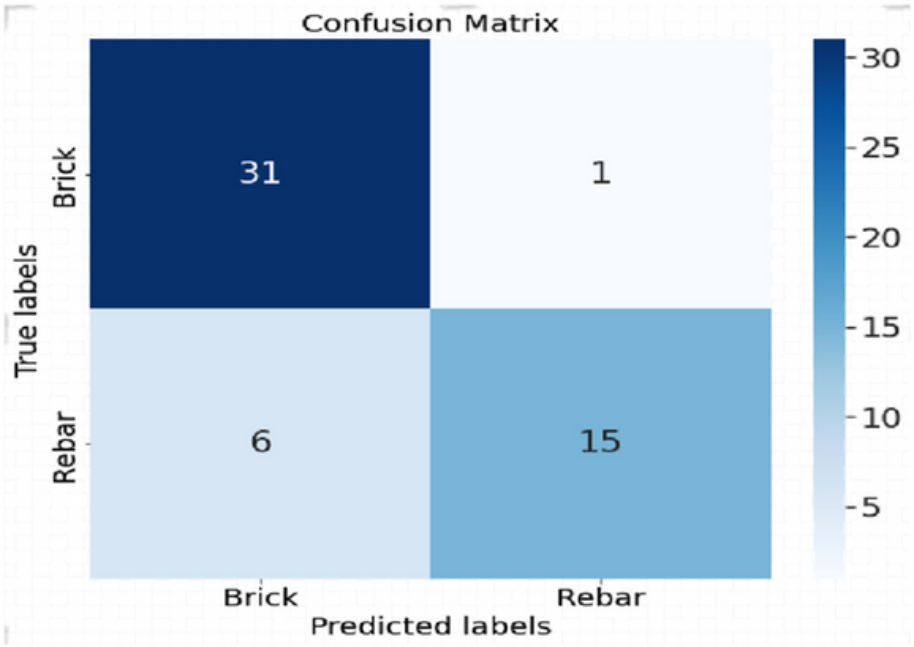
```
cnn.fit(x_train,y_train,epochs=10)
```

```
Epoch 1/10
7/7 [==============================] - 63s 9s/step - loss: 14.2292 - accuracy: 0.5613
Epoch 2/10
7/7 [==============================] - 56s 8s/step - loss: 1.1683 - accuracy: 0.5991
Epoch 3/10
7/7 [==============================] - 59s 8s/step - loss: 0.6597 - accuracy: 0.6226
Epoch 4/10
7/7 [==============================] - 55s 8s/step - loss: 0.5350 - accuracy: 0.8679
Epoch 5/10
7/7 [==============================] - 55s 8s/step - loss: 0.4277 - accuracy: 0.8585
Epoch 6/10
7/7 [==============================] - 54s 8s/step - loss: 0.3100 - accuracy: 0.8679
Epoch 7/10
7/7 [==============================] - 57s 8s/step - loss: 0.2278 - accuracy: 0.9104
Epoch 8/10
7/7 [==============================] - 55s 8s/step - loss: 0.1551 - accuracy: 0.9292
Epoch 9/10
7/7 [==============================] - 55s 8s/step - loss: 0.1836 - accuracy: 0.9245
Epoch 10/10
7/7 [==============================] - 55s 8s/step - loss: 0.1636 - accuracy: 0.9292
<keras.callbacks.History at 0x7fbc88f03d90>
```

Precision recall and accuracy

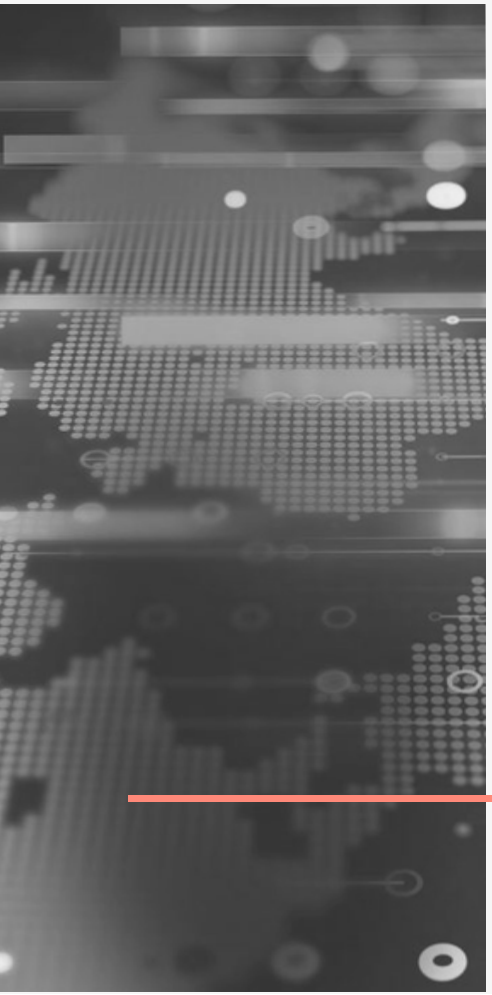
```
print(classification_report(y_pred,y_test))
```

```
              precision    recall  f1-score   support

           0       0.97      0.84      0.90        37
           1       0.71      0.94      0.81        16

    accuracy                           0.87        53
   macro avg       0.84      0.89      0.85        53
weighted avg       0.89      0.87      0.87        53
```



PROBLEMS FACED WHILE ADOPTING CNN

The CNN model was run without image augmentation, even though the training accuracy was very high, when test images were given, there was many misclassifications which is evident in the confusion matrix. Different image classification algorithms were yet to be tried out, and the next plausible algorithm seemed to be the YOLO algorithm.



YOLO

You only look once (YOLO) is a one-of-a-kind, real-time object recognition system. Object recognition/detection is an enhanced method of image classification where a neural network predicts the various objects present in an image and then points them out using a bounding box. Object detection can therefore be defined as the detection and localization of objects in an image which has predefined set of classes.

YOLO is tremendously quick and precise. Also, a simple tradeoff is possible between speed and accuracy simply by changing the size of the model. Compared to the approach taken by object detection algorithms before YOLO, which repurpose classifiers to perform detection, YOLO proposes the use of an end-to-end neural network that makes predictions of bounding boxes and class probabilities all at once. Following a fundamentally different approach to object detection, YOLO achieves state-of-the-art results beating other real-time object detection algorithms by a large margin.

OBJECT DETECTION

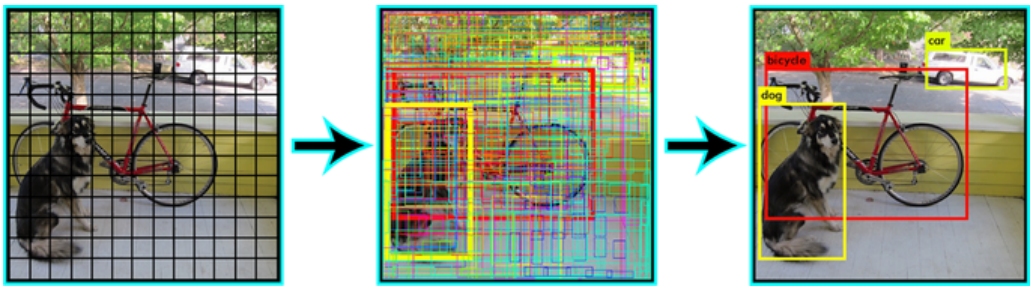
To explore the concept of object detection it's useful to begin with image classification. Image classification goes through levels of incremental complexity.

1. **Image classification** aims at assigning an image to one of a number of different categories (e.g., car, dog, cat, human, etc.), essentially answering the question "What is in this picture?". One image has only one category assigned to it.
2. **Object localization** allows us to locate our object in the image, so our question changes to "What is it and where it is?".
3. **Object detection** provides the tools for doing just that – finding all the objects in an image and drawing the so-called bounding boxes around them.

HOW DOES YOLO WORK?

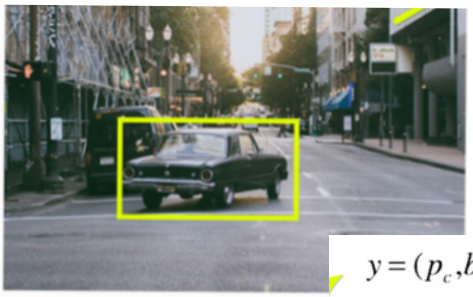
The YOLO algorithm works by dividing the image into N grids, each having an equal dimensional region of SxS. Each of these N grids is responsible for the detection and localization of the object it contains. Correspondingly, these grids predict B bounding box coordinates relative to their cell coordinates, along with the object label and probability of the object being present in the cell. This process greatly lowers the computation as both detection and recognition are handled by cells from the image.

But this approach brings forth a lot of duplicate predictions due to multiple cells predicting the same object with different bounding box predictions. YOLO makes use of Non-Maximal Suppression to deal with this issue. In Non-Maximal Suppression, YOLO suppresses all bounding boxes that have lower probability scores. YOLO achieves this by first looking at the probability scores associated with each decision and taking the largest one. Following this, it suppresses the bounding boxes having the largest Intersection over Union with the current high probability bounding box. This step is repeated till the final bounding boxes are obtained.

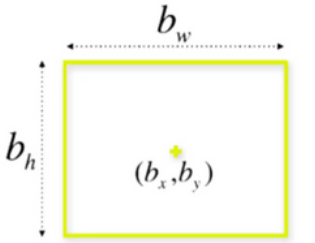


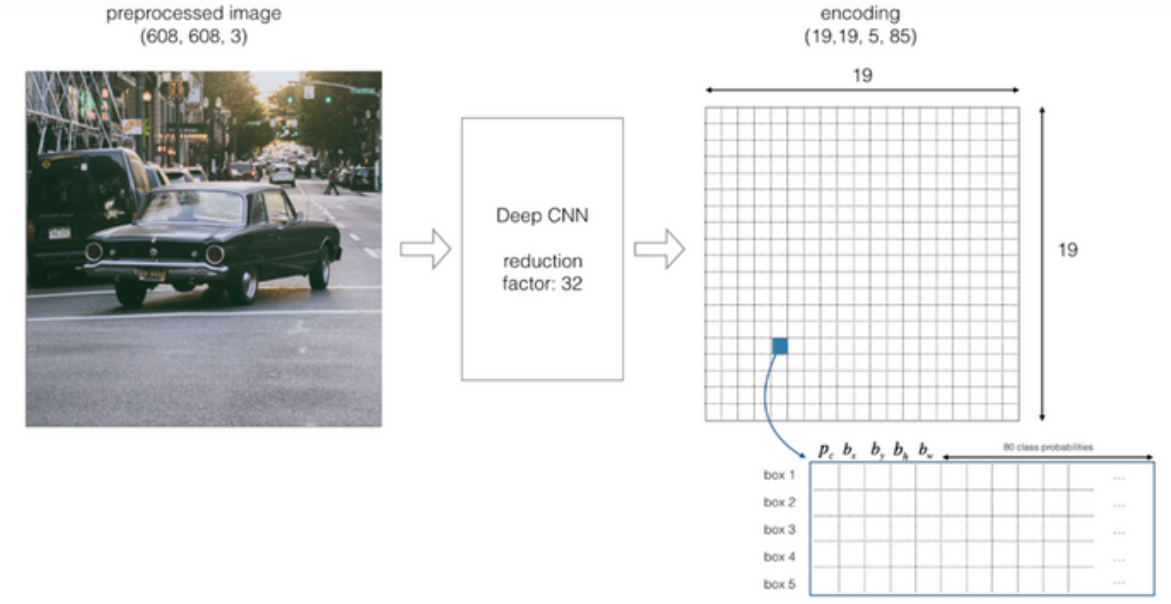
To understand the YOLO algorithm, it is necessary to establish what is actually being predicted. Ultimately, we aim to predict a class of an object and the bounding box specifying object location. Each bounding box can be described using four descriptors:

- center of a bounding box (bx,by)
- width (bw)
- height (bh)
- value cis corresponding to a class of an object



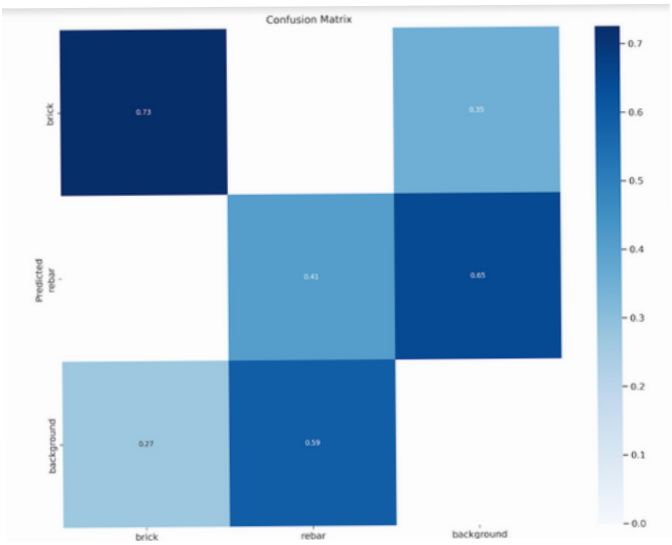
$$y = (p_c, b_x, b_y, b_h, b_w, c)$$





CONFUSION MATRIX & OBJECT DETECTION

Training images of bricks: 895
Training images of rebar: 805



YOLO CODE SNIPPETS

Cloning the YOLOV5 model from ultralytics GitHub

```
[ ]  !git clone https://github.com/ultralytics/yolov5
     %cd yolov5
     %pip install -qr requirements.txt

     import torch
     import utils
     display = utils.notebook_init()
```

Content provided in custom data.yaml file

```
train: ../train_data/images/train/
val: ../train_data/images/val/

# Classes
names:
  0: brick
  1: rebar
```

Code to run the YOLOV5 model for 300 epochs

```
!python train.py --img 500 --batch 10 --epochs 300 --data custom_data.yaml --weights yolov5s.pt --cache
```

LIMITATIONS OF YOLO

Although YOLO does seem to be the best algorithm to use if you have an object detection problem to solve, it comes with several limitations. YOLO struggles to detect and segregate small objects in images that appear in groups, as each grid is constrained to detect only a single object. Small objects that naturally come in groups, such as a line of ants, are therefore hard for YOLO to detect and localize.

For YOLO, the accuracy of the model is measured by how many bounding boxes it predicts correctly. In our model with sample images of bricks and rebar, YOLO was able to predict correctly whether a brick or rebar is present by showing a bounding box but it doesn't show all the bounding boxes as each grid is constrained to detect only one object. Because of this, the accuracy of model is low.

Also, we can only create rectangular bounding boxes while annotating images which cannot be tilted. This constraint takes lot of background noise into consideration and reduces accuracy (especially for rebar).

RANDOM FOREST

Training images of bricks: 483
Training images of rebar: 576

It is a collection of decision trees, one of the most powerful supervised classification algorithms. Many decision trees are built and the output received from each tree is collected and the majority is taken as the final output. Decision trees may suffer from the problem of overfitting, which will not be a problem in Random Forest because it takes opinions from many decision trees and the majority is given as the output. Random Forest is slower when compared to decision trees, which is very evident from the creation of many decision trees, even in our problem we had found the optimum number of decision trees to be 490 based on the predicting accuracy from the test data. We wanted to try out a classical classification algorithm before jumping on to complex CNN architecture like VGG19. We created augmented images and raised the dataset to 1059 images. As always initially, the image had to be converted to a NumPy array and normalized, while giving the input for Random Forest it was necessary to give the images as a 2-D array, so the NumPy array was resized from a 4-D array to a 2-D array. Finding the optimum no. of decision trees was one of the hyperparameters to be tuned. All other parameters in Random Forest were taken as such like the maximum features random forest uses before classifying the node, and the minimum number of leaves required to split an internal node. These parameters were used based on their default values. Only n-estimators were varied between 10 to 1000. Even XGBoost could have been used for finding the optimum number of decision trees. One important limitation in Random Forest and VGG19 was we could not use more than 1200 images for training and testing because the runtime kept crashing when the list of lists was converted into a NumPy array, so it was necessary for us to limit the number of images to 1059.

CODE SNIPPETS

Code to convert the image to an array, normalizing it, and converting it to a NumPy array

```
[ ]  for i in os.listdir('/content/drive/MyDrive/augimg1/rebar/'):
        correct_cat = 1
        #print('/content/drive/MyDrive/images/train/rebar/'+i)
        im = Image.open('/content/drive/MyDrive/augimg1/rebar/'+i)
        data = img_to_array(im)
        final_array.append(data/255)
        y.append(correct_cat)
     final_array=np.asarray(final_array)
```

Code to reshape the 4-D array to a 2-D array

```
[ ]  nsamples, nx, ny, nrgb = X_train.shape
     X_train2 = X_train.reshape((nsamples,nx*ny*nrgb))
```

Code to split the dataset in train and test accompanied by stratify and random state

```
[ ]  X_train, X_test, y_train, y_test = train_test_split(final_array, y, test_size=0.20,random_state=1, stratify=y)
```

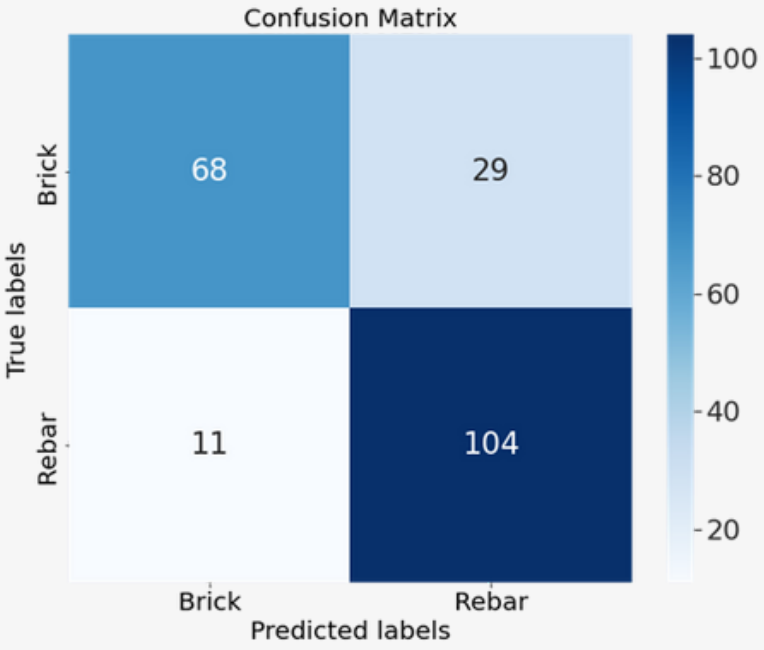
n-estimator was tuned by running it between 10 to 1000 and the accuracy in test data at each iteration was identified, we have cropped the images up to 490 because after that test accuracy hovered around 0.78. 490 was the optimized n-estimator (number of decision trees in Random Forest) which had an accuracy of 0.82 on the test dataset.

```
for i in range(10,1000,20):
  model=RandomForestClassifier(i)
  model.fit(X_train2,y_train)
  print('Accuracy in test dataset is for n estimator = '+str(i)+' is ',model.score(X_test2,y_test))
```

```
Accuracy in test dataset is for n estimator = 10 is  0.7641509433962265
Accuracy in test dataset is for n estimator = 30 is  0.7688679245283019
Accuracy in test dataset is for n estimator = 50 is  0.7783018867924528
Accuracy in test dataset is for n estimator = 70 is  0.7877358490566038
Accuracy in test dataset is for n estimator = 90 is  0.7830188679245284
Accuracy in test dataset is for n estimator = 110 is  0.7877358490566038
Accuracy in test dataset is for n estimator = 130 is  0.7971698113207547
Accuracy in test dataset is for n estimator = 150 is  0.7971698113207547
Accuracy in test dataset is for n estimator = 170 is  0.7971698113207547
Accuracy in test dataset is for n estimator = 190 is  0.7877358490566038
Accuracy in test dataset is for n estimator = 210 is  0.7971698113207547
Accuracy in test dataset is for n estimator = 230 is  0.7924528301886793
Accuracy in test dataset is for n estimator = 250 is  0.8066037735849056
Accuracy in test dataset is for n estimator = 270 is  0.7783018867924528
Accuracy in test dataset is for n estimator = 290 is  0.7877358490566038
Accuracy in test dataset is for n estimator = 310 is  0.7971698113207547
Accuracy in test dataset is for n estimator = 330 is  0.8113207547169812
Accuracy in test dataset is for n estimator = 350 is  0.8066037735849056
Accuracy in test dataset is for n estimator = 370 is  0.8113207547169812
Accuracy in test dataset is for n estimator = 390 is  0.7735849056603774
Accuracy in test dataset is for n estimator = 410 is  0.7971698113207547
Accuracy in test dataset is for n estimator = 430 is  0.7877358490566038
Accuracy in test dataset is for n estimator = 450 is  0.8113207547169812
Accuracy in test dataset is for n estimator = 470 is  0.8066037735849056
Accuracy in test dataset is for n estimator = 490 is  0.8254716981132075
```

CONFUSION MATRIX

A confusion Matrix is a good metric to quantify the performance of an Image classification problem. CNN had a higher training accuracy but the test accuracy was bad with test data, Random Forest had lesser training accuracy which eventually led to large misclassification in test data. Our next task is to reduce the misclassifications, so we would be trying out a well-tested image classification model VGG19, and we would be adopting a transfer learning approach. With a large number of convolution and max pooling structures, we feel VGG19 would be efficient in capturing the features of brick and rebar.

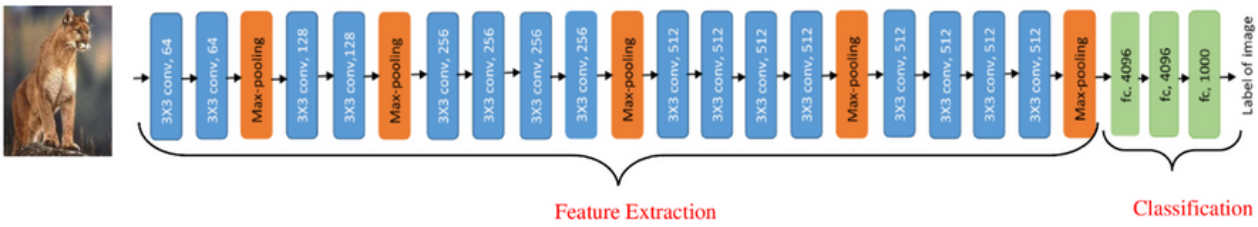


VGG19

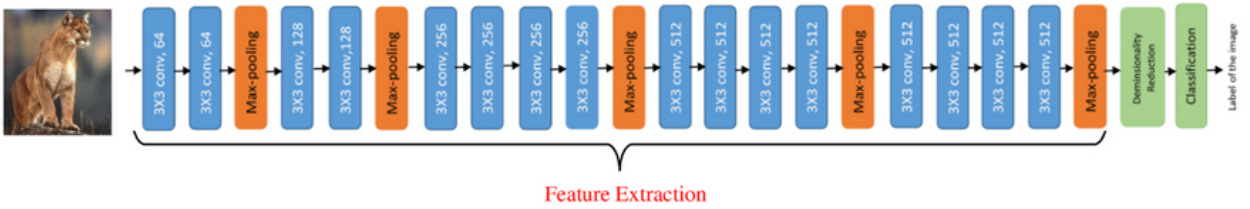
VISUAL GEOMETY GROUP

Training images of bricks: 483
Training images of rebar: 576

Further exploring the improved versions of CNN, we came across VGG which stands for Visual Geometry Group at the University of Oxford. VGGNet is a deep-learning algorithm with multiple layers. In Deep Learning, both feature extraction and classification are done simultaneously hence their performance is better in many traditional machine learning applications such as computer vision, speech identification, machine translation, and many more. One such important application is image recognition and classification, where VGGNet is specifically developed to serve this purpose. VGG19 algorithm is trained on the database ImageNet containing 14 million images belonging to 1000 categories. It has had many versions of architecture since its development. Here we intend to use VGG19, which consists of 19 layers out of which 16 are convolution layers, 3 fully connected layers (FC), 5 MaxPool layers, and 1 SoftMax layer. Figure below depicts the VGG19 architecture visually and it is followed by a detailed explanation.



(a) Architecture of VGG19 model



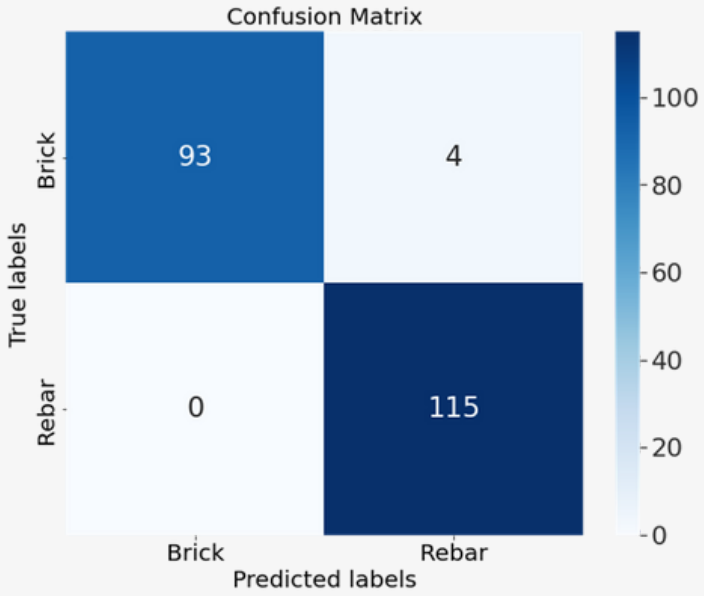
(b) Ensemble of deep feature extraction using VGG19 model and machine learning classification

VGG19 ARCHITECTURE

- The input for the VGG network is a 224*224 RGB image. In the pre-processing layer, each pixel value of the image is subtracted from the mean RGB value. And generally, in each convolution stage, kernels of 3*3 size with a stride size of 1 pixel are used and Max Pooling is carried out for 2*2 pixels with a stride size of 2 pixels.
- The first two layers are convolutional layers which use 64 filters and result in an image size of 224*224*64.
- Followed by is the max pooling layer, which reduces image size from 224*224*64 to 112*112*64.
- It is followed by 2 more convolution layers of 3*3 kernels and a 1-pixel stride with 128 filters, which results in a new image dimension of 112*112*128.
- Four more convolution layers are applied with 256 filters, followed by a max pooling layer, which again reduces the size of the image to 28*28*256.
- Following are two similar stacks, each with 4 convolution layers of 512 filters and separated by a max-pool layer which reduces the image size to 7*7*512.
- Flattened into 3 Connected (FC) layers of size 4096, 4096, & 1000 respectively.

ADVANTAGES
OF USING VGG-19

- Simple and easy to understand and execute.
- Much improved training speed compared to previous developments but is a bit slower than much of the very recent developments.
- Increase in non-linearity due to the increase in layers of smaller kernels
- High accuracy in prediction.



EXECUTION

The pre-processed (uniform size, image augmentation, and labeling) training data set of 1049 images were fed into the VGG19 model. The confusion matrix obtained using the test data is as follows,

The confusion matrix shows a prediction accuracy of 96% for bricks and 100% for rebars. Following are the brick images which are classified as rebar.

```
y_pred=model.predict(X_test)
y_pred=np.argmax(y_pred,axis=1)
print(classification_report(y_pred,y_test))
```

```
7/7 [==============================] - 31s 3s/step
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        93
           1       1.00      0.97      0.98       119

    accuracy                           0.98       212
   macro avg       0.98      0.98      0.98       212
weighted avg       0.98      0.98      0.98       212
```

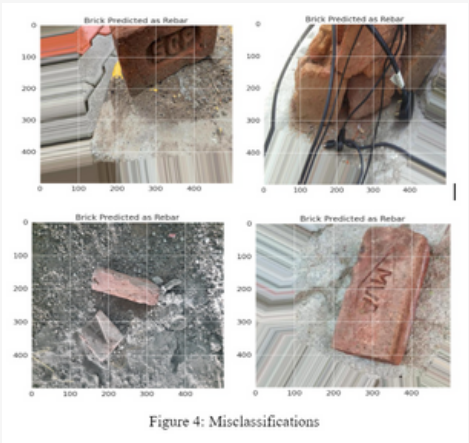


Figure 4: Misclassifications

CODE SNIPPETS

Following are some of the important code snippets and the explanation of the model.
VGG19 model is first initialized using the following piece of code. The size of the image was determined using the size statistics as 500*500. The weights for the model are taken from the predetermined weights from imagenet database through transfer learning. Transfer learning is to reduce the time consumed for training the model and also to reduce the GPU usage of our model.

```
[ ]  vgg = VGG19(input_shape = (500, 500, 3), weights = 'imagenet', include_top = False)
```

Then the dataset is split into training datasets and test datasets. We have considered 20% of the data as test data. “stratiy = y” is given to maintain the test and train data as constant, else for every run the test and train data will change.

```
X_train, X_test, y_train, y_test = train_test_split(final_array, y, test_size=0.20,random_state=1, stratify=y)
```

First image is opened, converted into an array and then normalized, to obtain the average pixel value and stored in the final array for training and validation.

```
[ ]  for i in os.listdir('/content/drive/MyDrive/images/val/brick/'):
        correct_cat = 0
        print('/content/drive/MyDrive/images/val/brick/'+i)
        im = Image.open('/content/drive/MyDrive/images/val/brick/'+i)
        data = img_to_array(im)
        final_array=np.append(final_array,data/255)
        y=np.append(y,correct_cat)
```

```
[ ]  for i in os.listdir('/content/drive/MyDrive/images/val/rebar/'):
        correct_cat = 1
        print('/content/drive/MyDrive/images/val/rebar/'+i)
        im = Image.open('/content/drive/MyDrive/images/val/rebar/'+i)
        data = img_to_array(im)
        final_array=np.append(final_array,data/255)
        y=np.append(y,correct_cat)
```

Other parameters are provided using the following code.

```
for layer in vgg.layers:
    layer.trainable = False

x = Flatten()(vgg.output)
prediction = Dense(2, activation='softmax')(x)

model = Model(inputs=vgg.input, outputs=prediction)
model.summary()
model.compile(
  loss='sparse_categorical_crossentropy',
  optimizer="adam",
  metrics=['accuracy']
)
```

When new images are to be predicted, it is opened using the following code, classified and the output is printed.

```
[ ]  a=[]
     im = Image.open('/content/brickdebris7.jfif')
     imResize = im.resize((500,500), Image.ANTIALIAS)
     data = img_to_array(imResize)
     a.append(data/255)
     a=np.array(a)
     y_pred=model.predict(a)
     print(y_pred)
```

Though VGG19 has classified and predicted with good efficiency, there is no localization of the images and no bounding box over the particular image.

For the training of VGG19, five epochs were used. The loss value and accuracy of each step of epochs is as shown below. As seen in the image the training time is also very less as we have used the weights from the pre-determined model trained using ImageNet database. “Early stop” is provided to stop the epoch automatically when the loss value for five consecutive epochs remains constant.

```
[25] history = model.fit(X_train, y_train, epochs = 5,callbacks=[early_stop],batch_size = 15,shuffle=True)

Epoch 1/5
57/57 [==============================] - ETA: 0s - loss: 0.9319 - accuracy: 0.8560WARNING:tensorflow:Early
57/57 [==============================] - 43s 475ms/step - loss: 0.9319 - accuracy: 0.8560
Epoch 2/5
57/57 [==============================] - ETA: 0s - loss: 0.0316 - accuracy: 0.9847WARNING:tensorflow:Early
57/57 [==============================] - 25s 437ms/step - loss: 0.0316 - accuracy: 0.9847
Epoch 3/5
57/57 [==============================] - ETA: 0s - loss: 0.0071 - accuracy: 0.9988WARNING:tensorflow:Early
57/57 [==============================] - 26s 459ms/step - loss: 0.0071 - accuracy: 0.9988
Epoch 4/5
57/57 [==============================] - ETA: 0s - loss: 0.0012 - accuracy: 1.0000WARNING:tensorflow:Early
57/57 [==============================] - 28s 494ms/step - loss: 0.0012 - accuracy: 1.0000
Epoch 5/5
57/57 [==============================] - ETA: 0s - loss: 4.8991e-04 - accuracy: 1.0000WARNING:tensorflow:
57/57 [==============================] - 27s 477ms/step - loss: 4.8991e-04 - accuracy: 1.0000
```

REFERENCES

1. Bansal, M., Kumar, M., Sachdeva, M. et al. Transfer learning for image classification using VGG19: Caltech-101 image data set. J Ambient Intell Human Comput (2021).
2. https://iq.opengenus.org/vgg19-architecture/ (last accessed on 12-10-2022)
3. https://viso.ai/deep-learning/vgg-very-deep-convolutional-networks/ (last accessed on 12-10-2022)
4. https://appsilon.com/object-detection-yolo-algorithm/(last accessed on 12-10-2022)
5. https://www.v7labs.com/blog/yolo-object-detection(last accessed on 13-10-2022)
6. https://pjreddie.com/darknet/yolo/(last accessed on 14-10-2022)
7. https://www.fastcompany.com/90721680/how-one-nyc-construction-company-saved-96-of-its-waste-from-the-landfill(last accessed on 12-10-2022)
8. Davis, P., Aziz, F., Newaz, M. T., Sher, W., & Simon, L. (2021). The classification of construction waste material using a deep convolutional neural network. Automation in Construction, 122, 103481. https://doi.org/10.1016/j.autcon.2020.103481
9. Zhang, Q., Yang, Q., Zhang, X., Bao, Q., Su, J., & Liu, X. (2021). Waste image classification based on transfer learning and convolutional neural network. Waste Management, 135, 150-157. https://doi.org/10.1016/j.wasman.2021.08.038
10. Jiang, P., Ergu, D., Liu, F., Cai, Y., & Ma, B. (2022). A Review of Yolo Algorithm Developments. Procedia Computer Science, 199, 1066-1073. https://doi.org/10.1016/j.procs.2022.01.135
11. Paymode, A. S., & Malode, V. B. (2022). Transfer Learning for Multi-Crop Leaf Disease Image Classification using Convolutional Neural Network VGG. Artificial Intelligence in Agriculture, 6, 23-33. https://doi.org/10.1016/j.aiia.2021.12.002
12. Alzubaidi, L., Zhang, J., Humaidi, A.J. et al. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. J Big Data 8, 53 (2021). https://doi.org/10.1186/s40537-021-00444-8
13. Basic CNN Architecture: Explaining 5 Layers of Convolutional Neural Network | upGrad blog
14. Convolutional Neural Network Definition | DeepAI
15. Albelwi S, Mahmood A. A Framework for Designing the Architectures of Deep Convolutional Neural Networks. Entropy. 2017; 19(6):242. https://doi.org/10.3390/e19060242