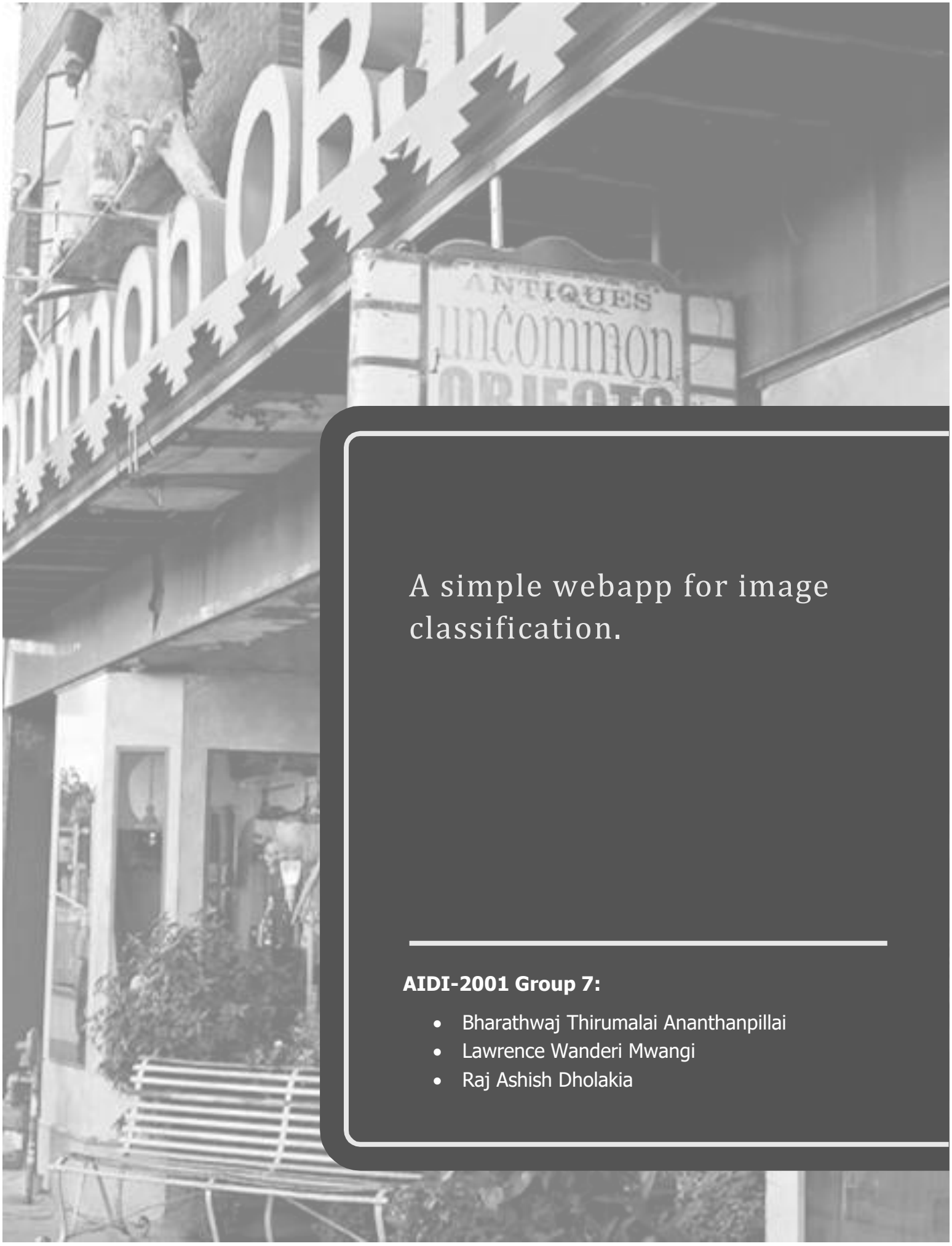


A simple webapp for image classification.

AIDI-2001 Group 7:

- Bharathwaj Thirumalai Ananthanpillai
- Lawrence Wanderi Mwangi
- Raj Ashish Dholakia

TABLE OF CONTENTS

1. Problem definition and Project Pitch

The field of artificial intelligence has applications in numerous parts of our lives, from manufacturing of the goods and services we consume to improving our security. With a significant growth in computer vision technology over the past five years, object detection and segmentation has become part of many applications being developed today. Object detection with the numerous large datasets, like COCO, has provided consistent and reliable computer vision models to be open-sourced. One area of computer vision that has not been widely explored is the ability to classify objects by their shape.

Shape detection for objects is an area that does not seem to have many real-world applications but is a topic that is important enough to be taught to children. Some of the niche applications of shape detection would be in manufacturing processes (for quality checks, etc.) and developing bionic arms. Understanding the shape of an object can be beneficial in those scenarios.

Therefore, for this project, we decided to take on the challenge of detecting 2D shapes of common and uncommon objects.

Why 2D shapes? Firstly, defining the shape of an object is not easy as it might sound. What would be the shape of pencil, window, bowl, or plate? Just look around you and identify at five objects that you and your neighbor will agree upon. What are some 3D shapes you can think of? It is difficult to have a conventional system for 3D shapes. For 2D shapes that is easier. Therefore, we decided to begin with 2D shape detection instead of 3D. The following shapes are considered:

- Circle
- Triangle
- Square
- Rectangle
- Pentagon
- Hexagon

Tricky: we can expect classifying a square and a rectangle to be the most challenging task.

Input: an image and object to be detected

Output: Shape prediction and confidence of prediction.

A. CHANGE FROM PROJECT PROPOSAL

In the project proposal, we had decided to use one of the datasets provided. However, after some exploratory data analysis and studying the performance of the models built on the dataset, we realized, we need to pivot.

We had two choices:

1. *We could simply the project by deploying a pre-trained object detection model (with a few-shot learning for shape detection). This would reduce the time we spend on building the model, hence a lower project completion time.*
2. *Second choice was to create a dataset to train a more reliable model, naturally, this was a more challenging path and would require ore effort.*

Regardless of the two paths, we had to make the change from the original plan as that did not provide us with the expected results.

Note: with more time and resources, we would have spent more time refining the dataset to improve the model's performance.

B. FUTURE SCOPE

There are several paths one can take to improve the project and take it a step further.

1. *Move from 2D to 3D shapes.*
2. *2D shape detection in real-world images.*
3. *3D shape detection with object detection.*
4. *Very ambitious: using shape detection in a simulation environment to test shape transformations.*

2. Exploratory Data Analysis

The dataset was acquired through a simple google search of

- "[shape]"
- "[shape] real-life objects"

A. Examples

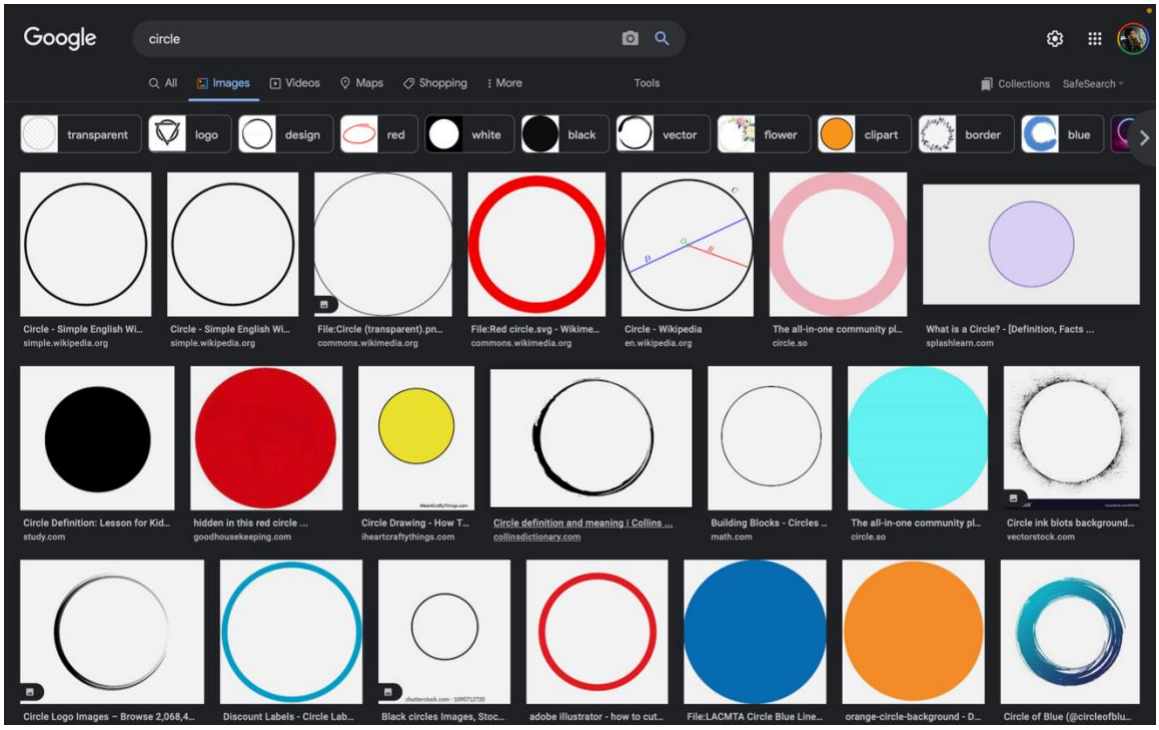


Figure 1: Google Image Search for "circle".

After downloading the images, all the downloads were verified. If the image opened and was in the right the format, the image was approved.

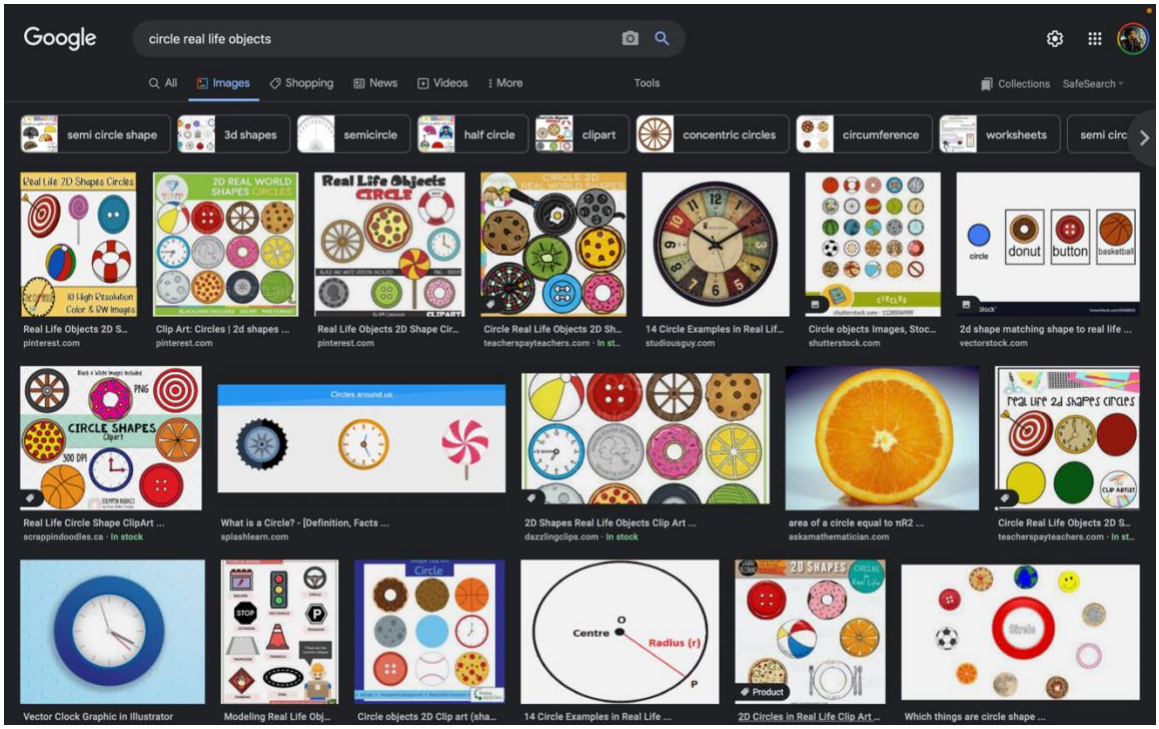


Figure 2: Google Image Search for "circle real life objects".

Following is the number of images available for training after the verification process.

```
# training images for each class
for shape, files in train_shape_files.items():
    print(f"{shape}: \t{len(files)} images")
```

```
circle:         232 images
triangle:       360 images
square:         250 images
rectangle:      255 images
pentagon:       321 images
hexagon:        233 images
```

```
# All the images for each class
for shape, files in all_shape_files.items():
    print(f"{shape}: \t{len(files)} images")
```

```
circle:         331 images
triangle:       514 images
square:         357 images
rectangle:      365 images
pentagon:       459 images
hexagon:        333 images
```

Figure 3: Total # of examples & # of training examples (images) by class.

Note: more manual cleaning of the images is required, but due to the limited time and resources, we were not able to thoroughly clean the dataset.

Preliminary or minor deletion of irrelevant images was carried out once the dataset was downloaded.

```
fig, ax = plt.subplots(figsize=(7,7))

ax.bar(x=list(shapes_images.keys()), height=list(shapes_images.values()));

plt.xlabel('Shape', fontdict={'size': 14, 'weight': 'bold'});
plt.ylabel('Number of images', fontdict={'size': 14, 'weight': 'bold'});
plt.title('Training Images per Class', fontdict={'size': 16, 'weight': 'bold'});
```

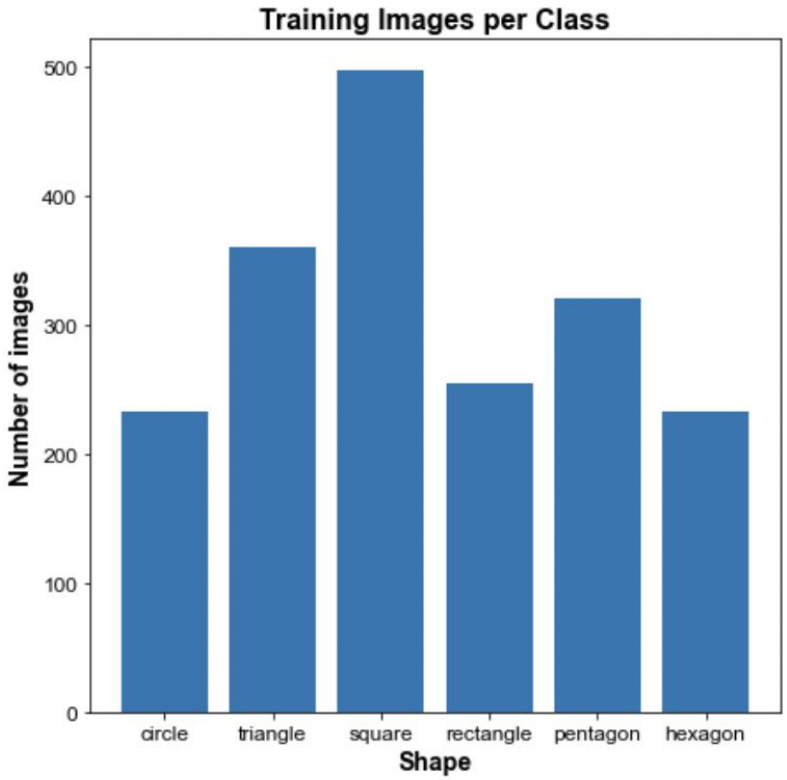


Figure 4: Bar plot of the # of Training examples (images) by class.

The number of training images for squares are the highest.

Additionally, there is minimal balance between the classes, with circles have the least number of examples.

More balance can be created in the dataset by adding more samples to the dataset or using data augmentation to introduce a variation of existing images.

3. Solution Development

The solution to the problem of shape detection can be split into the three main parts:

1. Data Collection
2. Model Training and Evaluation
3. Model Deployment

We will note go through each of the sections in detail.

A. Data Collection

The goal of the data collection process is to get enough images to train a reliable deep neural network to predict the shape of a given object.

Note: For the most part, we followed the tutorial by *Adrian Rosebrock*.

This was split into the following processes:

- **Image Search**: Searching for the image on Google Images.
- **Save URLs (JavaScript)**: Saving the list of URLs of the images to a .txt file using JavaScript Console on Google Chrome.
- **Download Images (Python)**: Accessing the .txt files in a Python Environment and downloading the images using the `requests` python package. Use `uuid` to give each image a unique ID.
- **Image Verification**: After downloading all available images, verify the downloaded image files. The verification process opens the image using `opencv`s open image method, if the image does not open, it is **marked corrupt and deleted**.
- **Manual check**: Visual verification by human of the images is carried out, where irrelevant images are deleted. NOTE: Due to the number of images we gathered (about 2800), we were not able to go through all the images – hence the performance of the model can be easily improved by focusing more on cleaning the dataset.

```
shapes_images = {shape: len(files) for shape, files in shape_files.items()}
shapes_images
```

```
{'circle': 233,
 'triangle': 360,
 'square': 497,
 'rectangle': 255,
 'pentagon': 321,
 'hexagon': 233}
```

```
# Total number of images in the dataset
sum(shapes_images.values())
```

```
1899
```

Figure 5: Total # of Training images

All the steps above are coded into functions for ease of understanding and modular use-case. This is due to the PEP8 standard of python we are following throughout this project.

B. Model Training and Evaluation

We used Vgg16 model for training the images as shown in Figure 6 due to its deep extraction of image features. And mostly we used only two shapes – Circle, & Square to check whether the model works properly or not. So Binary Cross entropy loss function is used, and we are using ADAM optimizer so that it reaches minimum value of Loss.

```
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dropout
from tensorflow.keras.optimizers import SGD

#VGG16 Architecture used for Convolution of images

base_model=tf.keras.applications.vgg16.VGG16(
    include_top=False,
    weights=None,
    input_tensor=None,
    input_shape=(224,224,3),
    pooling=None
)

for layer in base_model.layers:
        layer.trainable = False

x=base_model.output

model=Flatten()(x)
model=Dense(units=5096,activation="relu")(model)
model=Dropout(0.5)(model)
model=Dense(units=5096,activation="relu")(model)
output=Dense(units=1, activation="sigmoid")(model)

model = keras.models.Model(inputs=base_model.input, outputs = output)

from tensorflow.keras.optimizers import Adam
opt = Adam(lr=0.001)
model.compile(optimizer=opt, loss="binary_crossentropy", metrics=['accuracy'])
model.summary()
```

Figure 6: VGG16 Model

Like one of our teammates, I (Bharathwaj) download images from google due to less amount of datasets for shape classifications when compared to object datasets.

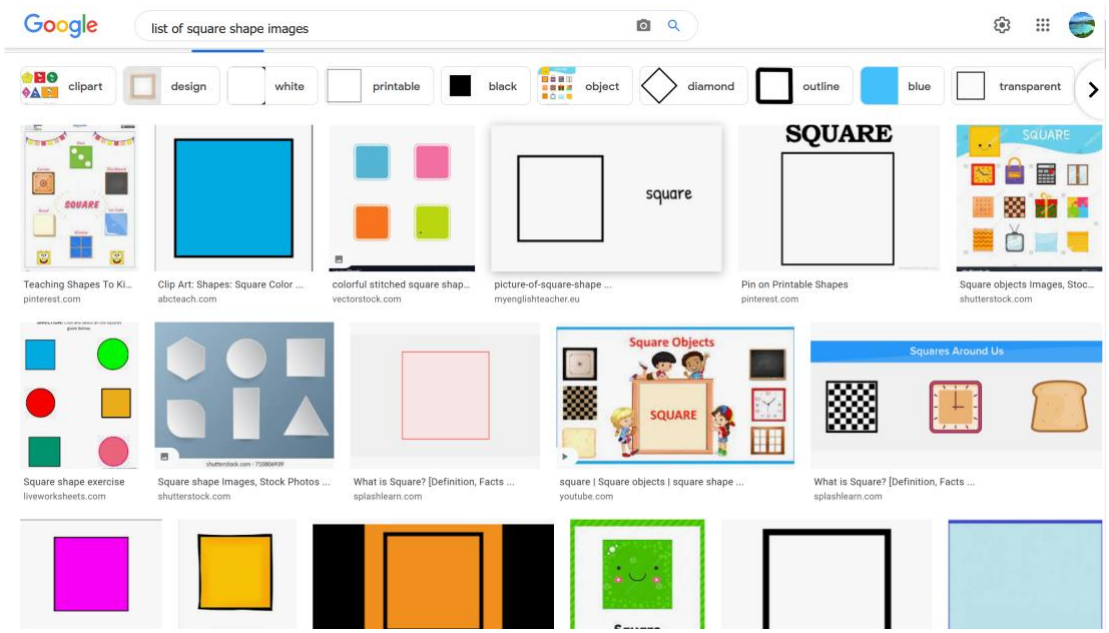


Figure 7: Square Images

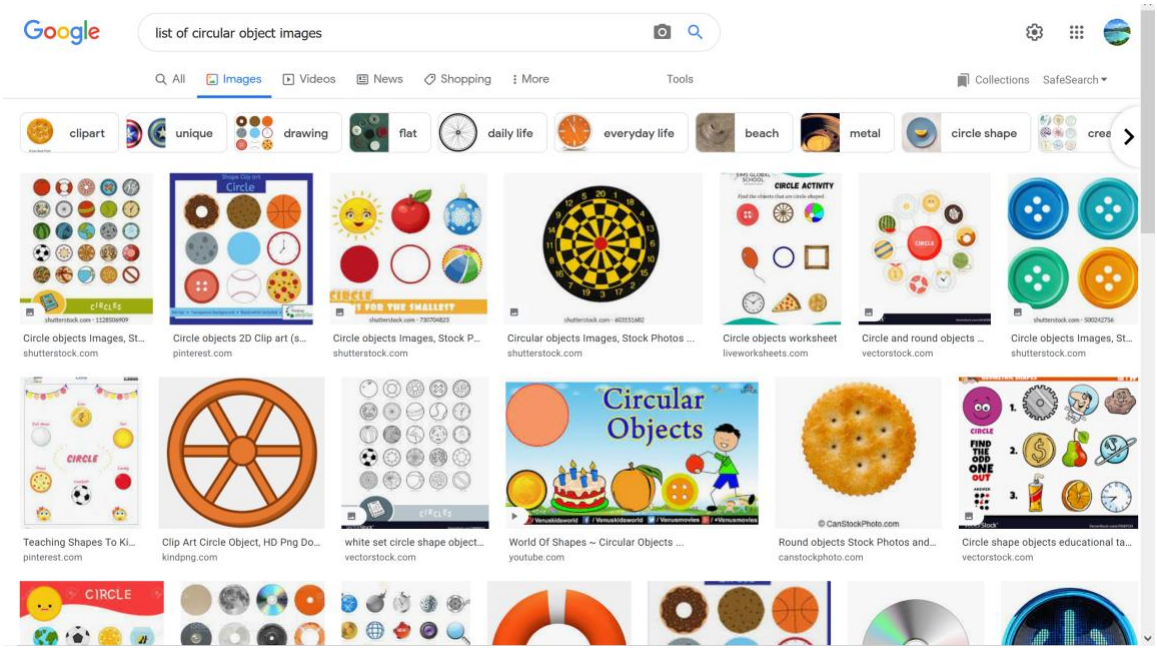


Figure 8: Circle Images

From google, 37 images were collected for both Circle and Square. All circle images were augmented in such that they are rotated of about 15 degree, zoom range of about 0.1 and so on which is shown in Figure below

```
augmentation=0
while augmentation < 3:

    circleTrainDatagen = ImageDataGenerator(rotation_range=15,zoom_range=0.1,rescale=1./255)
    CircleTrainPath="Shape Dataset/Train/Circle/"
    circleTrain = os.listdir(CircleTrainPath)

    for index, image in enumerate(circleTrain):          # for loop for training images of Circle.
        filename = image.split('.')
        if(filename[1] == 'png' or filename[1] == 'jpg' or filename[1] == 'jpeg'):
            # load the image using split concept.
            img = load_img(CircleTrainPath+filename[0]+'.'+filename[1])
            # convert to numpy array
            data = img_to_array(img)
            # expand dimension to one sample
            circleTrainSamples = expand_dims(data, axis=0)

            circleTrainIT = circleTrainDatagen.flow(circleTrainSamples, batch_size=32,save_to_dir= CircleTrainPath,
                                save_prefix="CIRCLE",
                                save_format='jpg')     #Using datagen.flow, every image is flipped horizontally,
                                                       #rotated at certain angle and finally stored in a specified directory.
            batch = circleTrainIT.next()

        #Same concept follows for test images of Square shown below
```

Figure 8: Circle Image Generation

Same followed for Square shape also

```
    squareValidationDatagen = ImageDataGenerator(zoom_range=0.1,rescale=1./255)
    SquareValidationPath="Shape Dataset/Validation/Square/"
    squareValidation = os.listdir(SquareTrainPath)

    for index, image in enumerate(squareTrain):
        filename = image.split('.')
        if(filename[1] == 'png' or filename[1] == 'jpg' or filename[1] == 'jpeg'):
            img = load_img(SquareTrainPath+filename[0]+'.'+filename[1])
            data = img_to_array(img)
            squareValidationSamples = expand_dims(data, axis=0)


            squareValidationIt = squareValidationDatagen.flow(squareValidationSamples, batch_size=32,save_to_dir= SquareValid
                                save_prefix="SQUARE",
                                save_format='jpg')     #Using datagen.flow, every image is flipped horizontally,
                                                       #rotated at certain angle and finally stored in a specified directory.
            batch = squareValidationIt.next()

    augmentation+=1
```

Figure 9: Square Image Generation

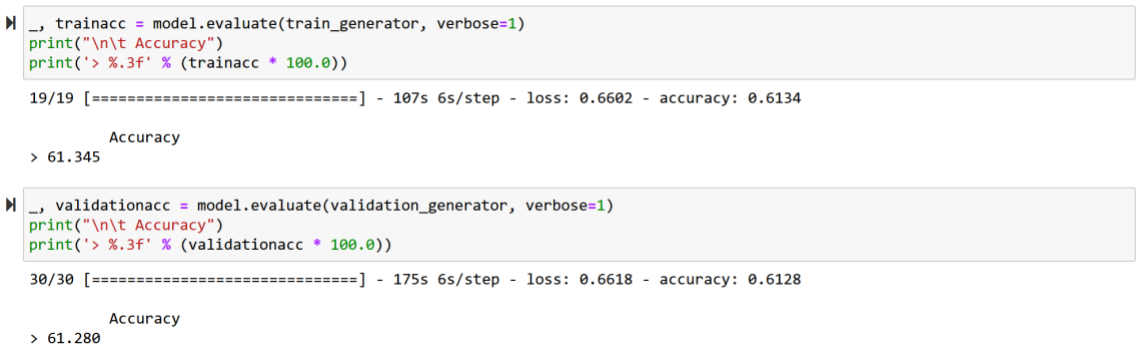
These steps resulted in increase in quantity of images in dataset as shown below…….

```
trainDatagen = ImageDataGenerator(rescale=1./255)

testDatagen = ImageDataGenerator(rescale=1./255)

train_generator = trainDatagen.flow_from_directory(
    "Shape Dataset/Train",
    target_size=(224,224),
    batch_size=32,
    class_mode='binary')

validation_generator = testDatagen.flow_from_directory(
    "Shape Dataset/Validation",
    target_size=(224,224),
    batch_size=32,
    class_mode='binary')
```

```
Found 595 images belonging to 2 classes.
Found 953 images belonging to 2 classes.
```

Figure 10: No of images after Data Augmentation

.....and using Learning rate of about 0.001 for Adam Optimizer, Sigmoid Activation function, Batch size of 32 and Epoch of 50, we obtained an accuracy of about 61.345% in training set and 61.280% in validation set.



```
_, trainacc = model.evaluate(train_generator, verbose=1)
print("\n\t Accuracy")
print('> %.3f' % (trainacc * 100.0))
```

```
19/19 [==============================] - 107s 6s/step - loss: 0.6602 - accuracy: 0.6134

         Accuracy
> 61.345
```

```
_, validationacc = model.evaluate(validation_generator, verbose=1)
print("\n\t Accuracy")
print('> %.3f' % (validationacc * 100.0))
```

```
30/30 [==============================] - 175s 6s/step - loss: 0.6618 - accuracy: 0.6128

         Accuracy
> 61.280
```

Figure 11: Accuracy of the model

As already stated, lack of accuracy is due to lack of images and lack of data cleaning. Along with this, Epoch values & learning rate variation also plays a major role.

C. Model Deployment

i. Flask Backend

Pre-processes the image provided by the user, loads the model, and returns the model's prediction.

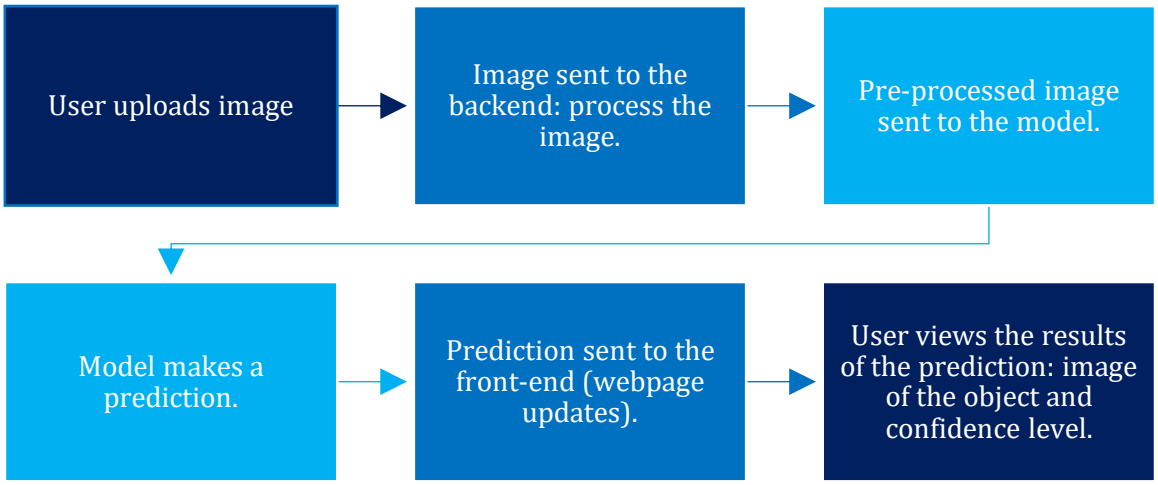
ii. HTML/CSS Frontend

Creates the webpage, manages the data transfer to the backend and the simple UI provides the user a way to interact with the model. Some points other noteworthy points:

1. *upload limits (image size – 5MB)*
2. *Image formats accepted: .jpg or .png*

iii. Cloud Deployment

Heroku was used to deploy the webpage to the cloud. A PROCFILE was created to define the environment for the container deployment.



4. Solution Deployment

The application was deployed on Heroku

The solution was deployed on Heroku. Two files are key to create a container and deploy the webapp on Heroku. Firstly, the environment needs to be defined and that is done using the requirements.txt file.

The requirements.txt file is supported by the PROCFILE to determine the build and action to get the application started.

The PROCFILE along with the wsgi.py file are responsible to get the webapp to run once all the files have been uploaded to Heroku's cloud.

5. Coding Standard

We have attempted to follow the PEP8 python coding style guide to the best of our ability.

PEP8 Python Style Guide

Two exceptions were made to the PEP8 Python Style Guide:

1. *Docstrings for functions: Google Python Style Guide provided better way to describe the functions' purpose, its arguments and return variables.*
2. *Tabs instead of 4 spaces for indentation: I believe this to be a personal preference, however, the first answer on this question[1] on StackOverflow reflects my opinion.*

Google Python Style Guide

[1] "Why does Python pep-8 strongly recommend spaces over tabs for indentation?"

6. References

Link to GitHub Repository with the code used in this report.

https://github.com/mwangidyce/aidi_2004

Some Google Image Searches:

"list of circular object images"

"list of square shape images"

David Carty, April 2021, ML Algorithms and data splits
https://www.applause.com/blog/training-data-validation-data-vs-test-data

Jason Brownlee PhD, June 2016, Dropout Regularization in Deep Learning Models With Keras

https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/

Getty Images, 2022, Photos of Circular Objects

https://www.gettyimages.ca/photos/circular-objects