CODING CHALLENGE 2

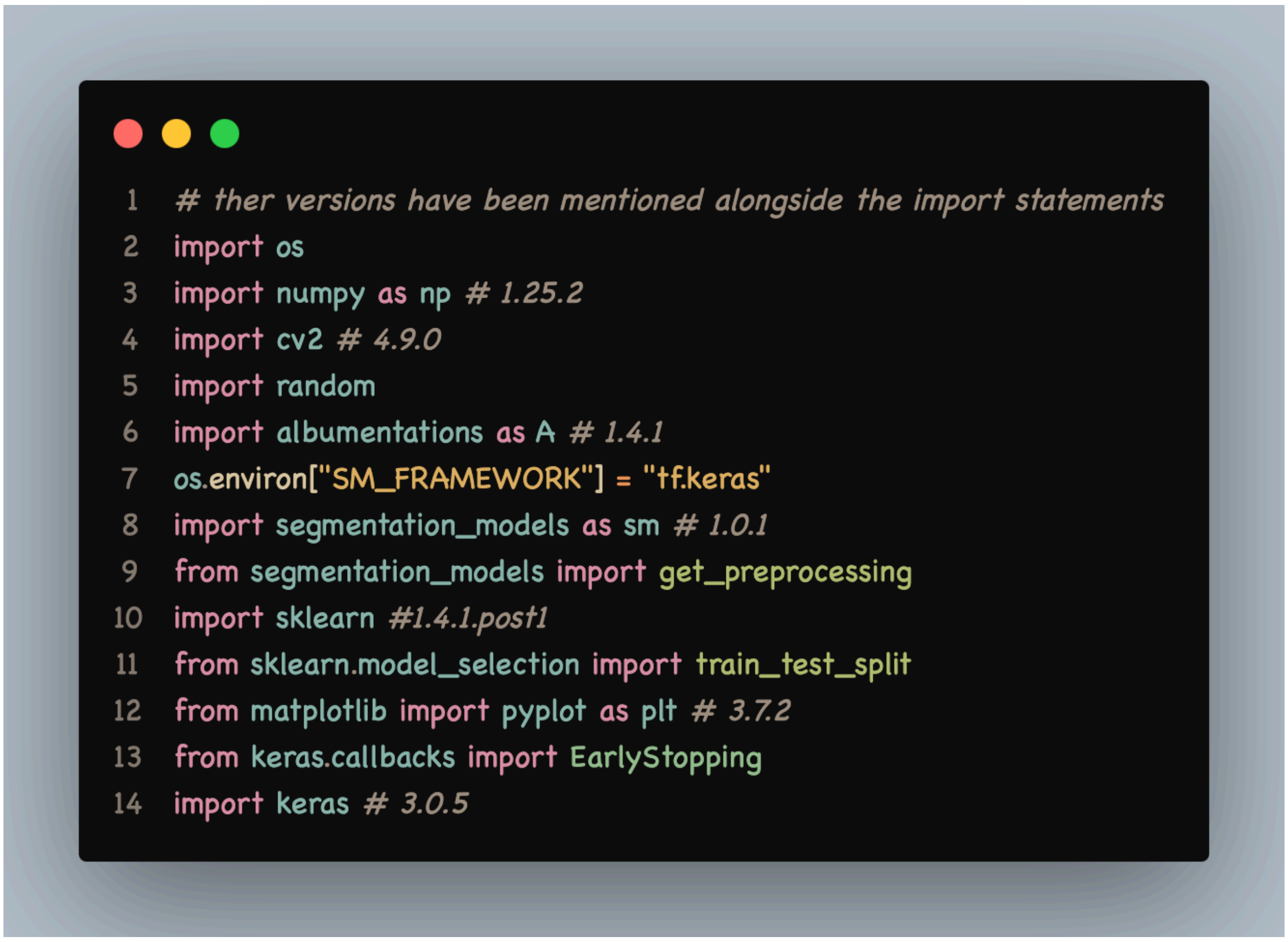
STONES SEGMENTATION

Akshat Sharma
AKSHATSHARMA8304@GMAIL.COM

INDEX

1. LIBRARIES AND MODULES REQUIRED

- OS
- Numpy
- OpenCV
- Random
- Albumentations
- Segmentation_models
- Scikit Learn
- Matplotlib
- Keras

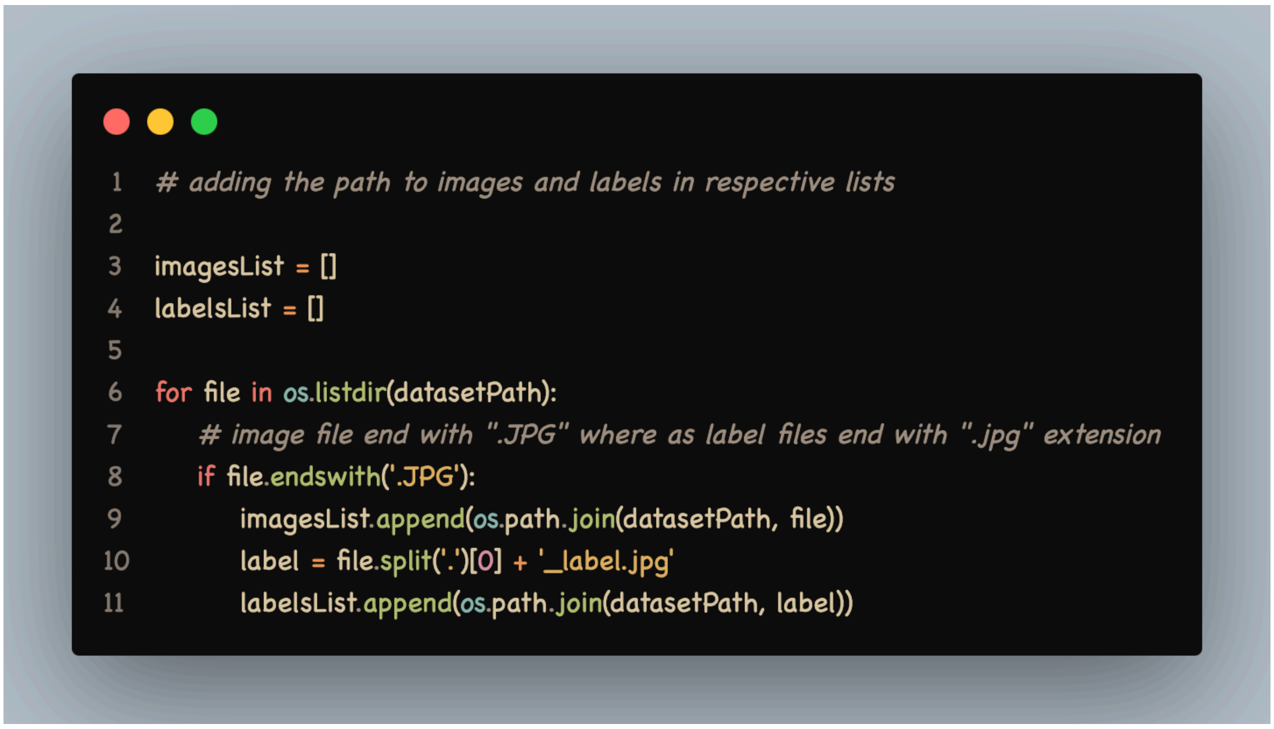


```python
# ther versions have been mentioned alongside the import statements
import os
import numpy as np # 1.25.2
import cv2 # 4.9.0
import random
import albumentations as A # 1.4.1
os.environ["SM_FRAMEWORK"] = "tf.keras"
import segmentation_models as sm # 1.0.1
from segmentation_models import get_preprocessing
import sklearn #1.4.1.post1
from sklearn.model_selection import train_test_split
from matplotlib import pyplot as plt # 3.7.2
from keras.callbacks import EarlyStopping
import keras # 3.0.5
```

2. DATA PREPARATION

- Total images provided (40) : 20 inputs and 20 labels

- Observations : input images have extension ".JPG" i.e uppercase and label images have extension ".jpg" lowercase. This make it easy for us to store the paths of the inputs and labels separately.



```python
# adding the path to images and labels in respective lists

imagesList = []
labelsList = []

for file in os.listdir(datasetPath):
    # image file end with ".JPG" where as label files end with ".jpg" extension
    if file.endswith('.JPG'):
        imagesList.append(os.path.join(datasetPath, file))
        label = file.split('.')[0] + '_label.jpg'
        labelsList.append(os.path.join(datasetPath, label))
```

3. DATA AUGMENTATION

- Since we only have 20 images and 20 labels, these are not enough to build a decent model that can accurately segment the stones from the background. In order to get more images, we will use the concept of augmentation.

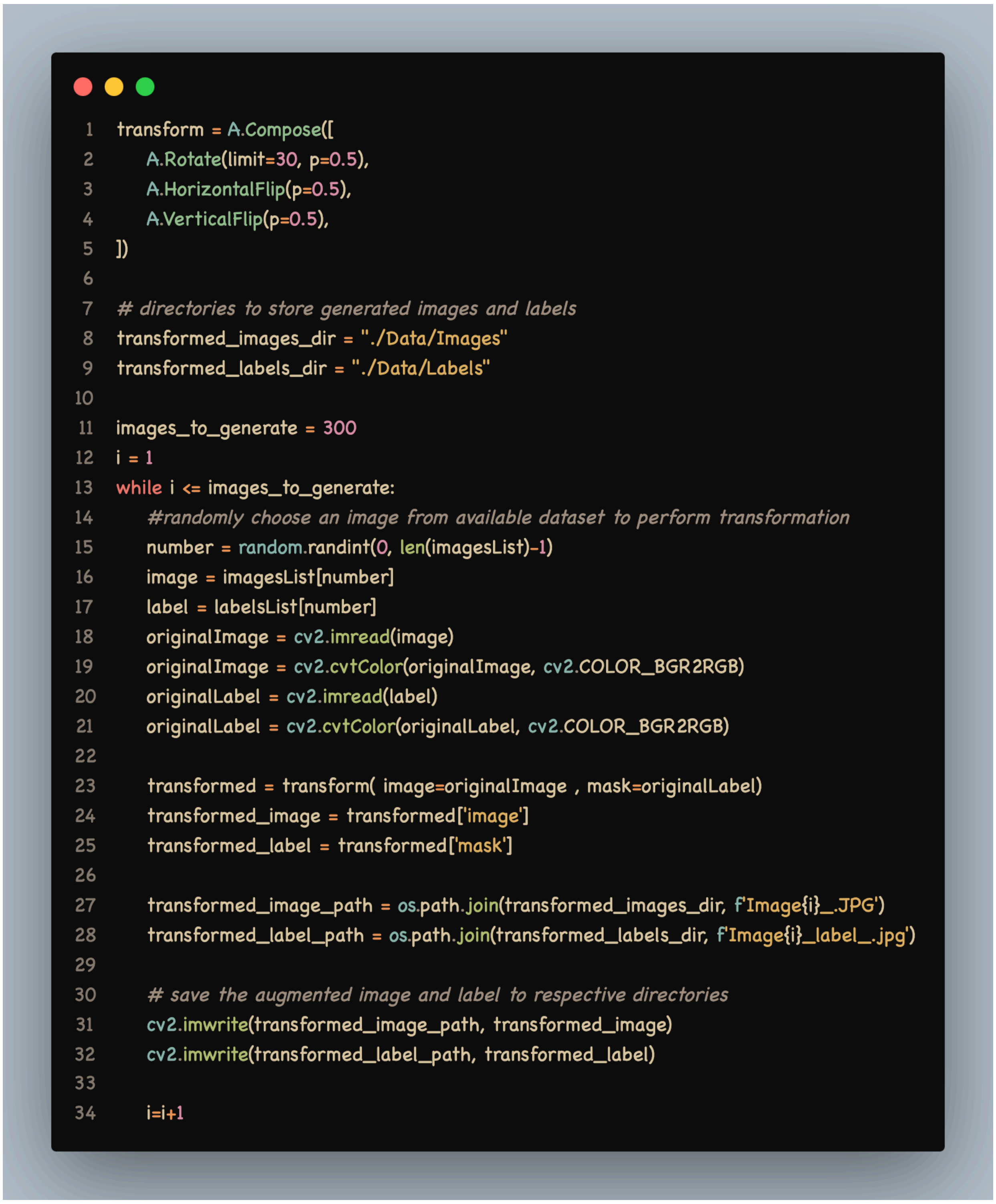
- Augmentation lets us generate more images from existing images by performing some transformations on them.

- We will use the albumentations library to create our augmented images.

- The number of augmented images should be close to ten times the images we have originally. Therefore we will generate 300 images for inputs and labels respectively.

- We are generating RGB images for both the inputs and labels (this will be changed later)

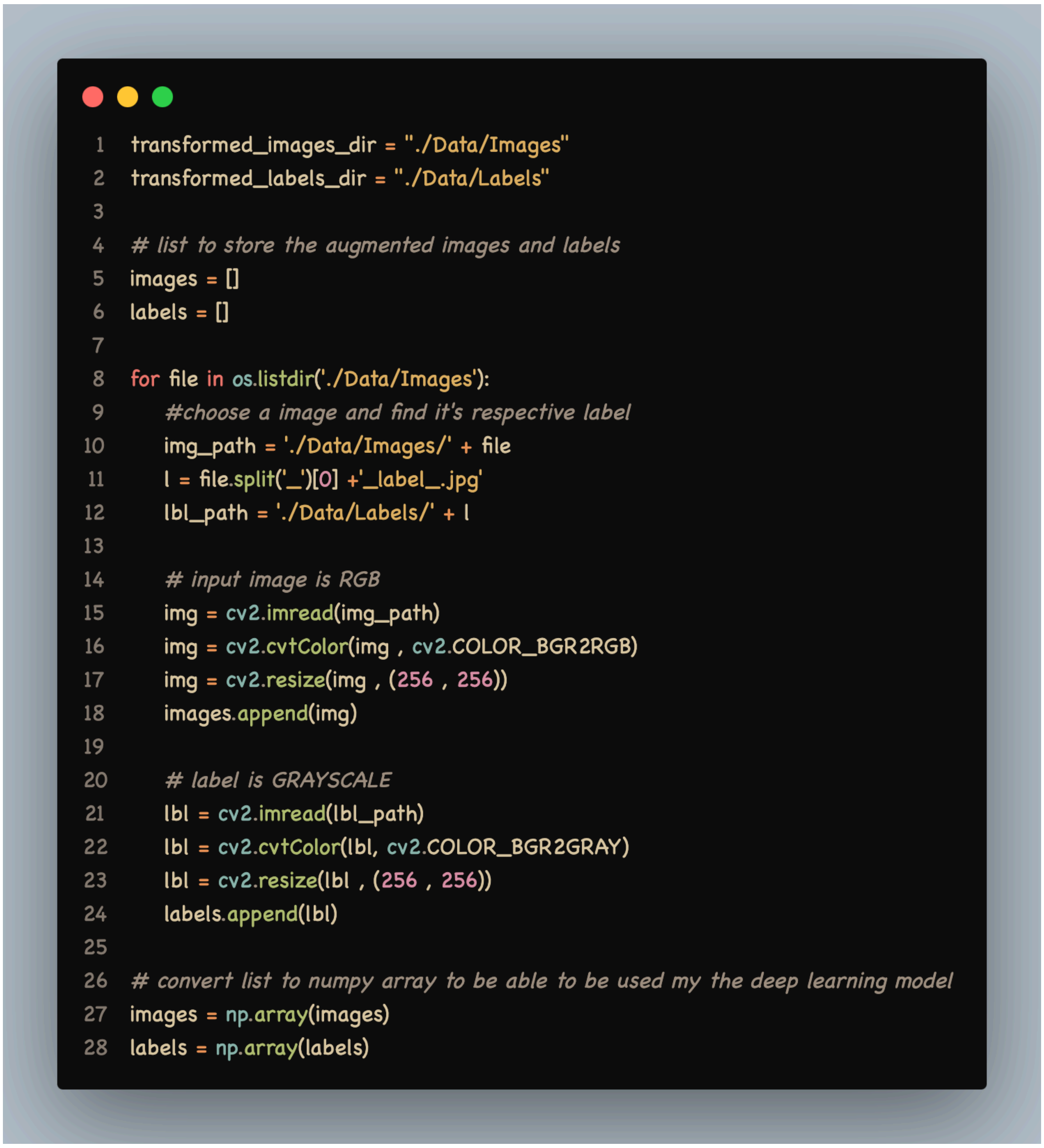
- The augmented images are stored in their respective folders



```python
transform = A.Compose([
    A.Rotate(limit=30, p=0.5),
    A.HorizontalFlip(p=0.5),
    A.VerticalFlip(p=0.5),
])

# directories to store generated images and labels
transformed_images_dir = "./Data/Images"
transformed_labels_dir = "./Data/Labels"

images_to_generate = 300
i = 1
while i <= images_to_generate:
    #randomly choose an image from available dataset to perform transformation
    number = random.randint(0, len(imagesList)-1)
    image = imagesList[number]
    label = labelsList[number]
    originalImage = cv2.imread(image)
    originalImage = cv2.cvtColor(originalImage, cv2.COLOR_BGR2RGB)
    originalLabel = cv2.imread(label)
    originalLabel = cv2.cvtColor(originalLabel, cv2.COLOR_BGR2RGB)

    transformed = transform( image=originalImage , mask=originalLabel)
    transformed_image = transformed['image']
    transformed_label = transformed['mask']

    transformed_image_path = os.path.join(transformed_images_dir, f'Image{i}_.JPG')
    transformed_label_path = os.path.join(transformed_labels_dir, f'Image{i}_label_.jpg')

    # save the augmented image and label to respective directories
    cv2.imwrite(transformed_image_path, transformed_image)
    cv2.imwrite(transformed_label_path, transformed_label)

    i=i+1
```

4 . DATA PROCESSING

- The images we have very large in size. Their shape is (3456, 4806, 3). We cannot use such large images in our model as it can cause computation challenges. There we will convert our images to (256 , 256 ,3)
- We can also observe that the label is essentially a black and white image therefore we can convert our label images into grayscale(single channel).
- As we cannot directly use images in a model, we also need to create arrays that contain images in the form of numpy arrays.



```python
transformed_images_dir = "./Data/Images"
transformed_labels_dir = "./Data/Labels"

# list to store the augmented images and labels
images = []
labels = []

for file in os.listdir('./Data/Images'):
    #choose a image and find it's respective label
    img_path = './Data/Images/' + file
    l = file.split('_')[0] +'_label_.jpg'
    lbl_path = './Data/Labels/' + l

    # input image is RGB
    img = cv2.imread(img_path)
    img = cv2.cvtColor(img , cv2.COLOR_BGR2RGB)
    img = cv2.resize(img , (256 , 256))
    images.append(img)

    # label is GRAYSCALE
    lbl = cv2.imread(lbl_path)
    lbl = cv2.cvtColor(lbl, cv2.COLOR_BGR2GRAY)
    lbl = cv2.resize(lbl , (256 , 256))
    labels.append(lbl)

# convert list to numpy array to be able to be used my the deep learning model
images = np.array(images)
labels = np.array(labels)
```

5. MODEL PREREQUISITES AND ARCHITECTURE

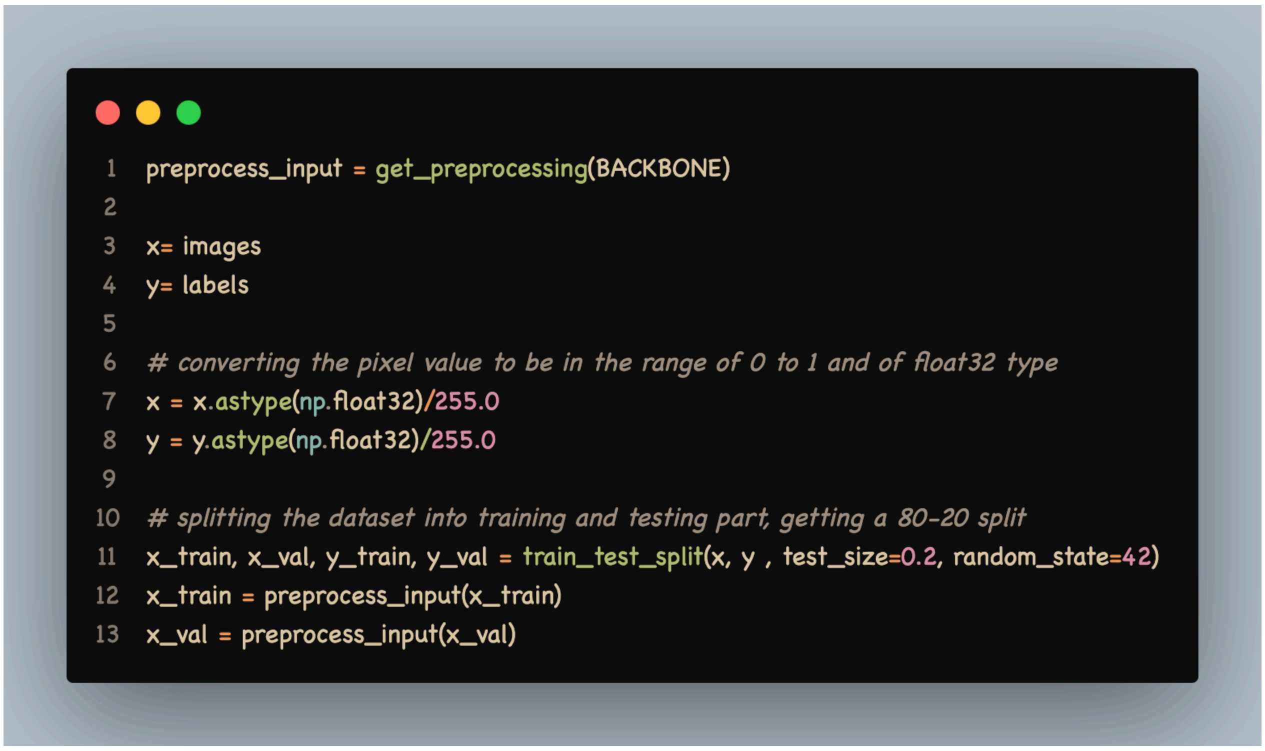
- Since we are dealing with an images segmentation problem here, the first choice of architecture would be Unet. It's encoder-decoder format is highly advantageous for segmentation. Also it is quite efficient in making use of augmented data.

- The ability of Unet to be trained in a full front to back manner makes it a advantageous choice since we can easily train our model without any additional preprocessing steps.

- For the backbone, of all the available options "resnet18" is being used. Since our segmentation problem is not very complex, we need a model that provides a balance between depth and complexity. The "resnet18" model also comes pre-trained on the "imagenet" dataset which makes it possible for us to use pre trained encoder weights for our model.

- For better computation we will convert the pixel value of our dataset into a range of 0 to 1 and in float32 dtype.

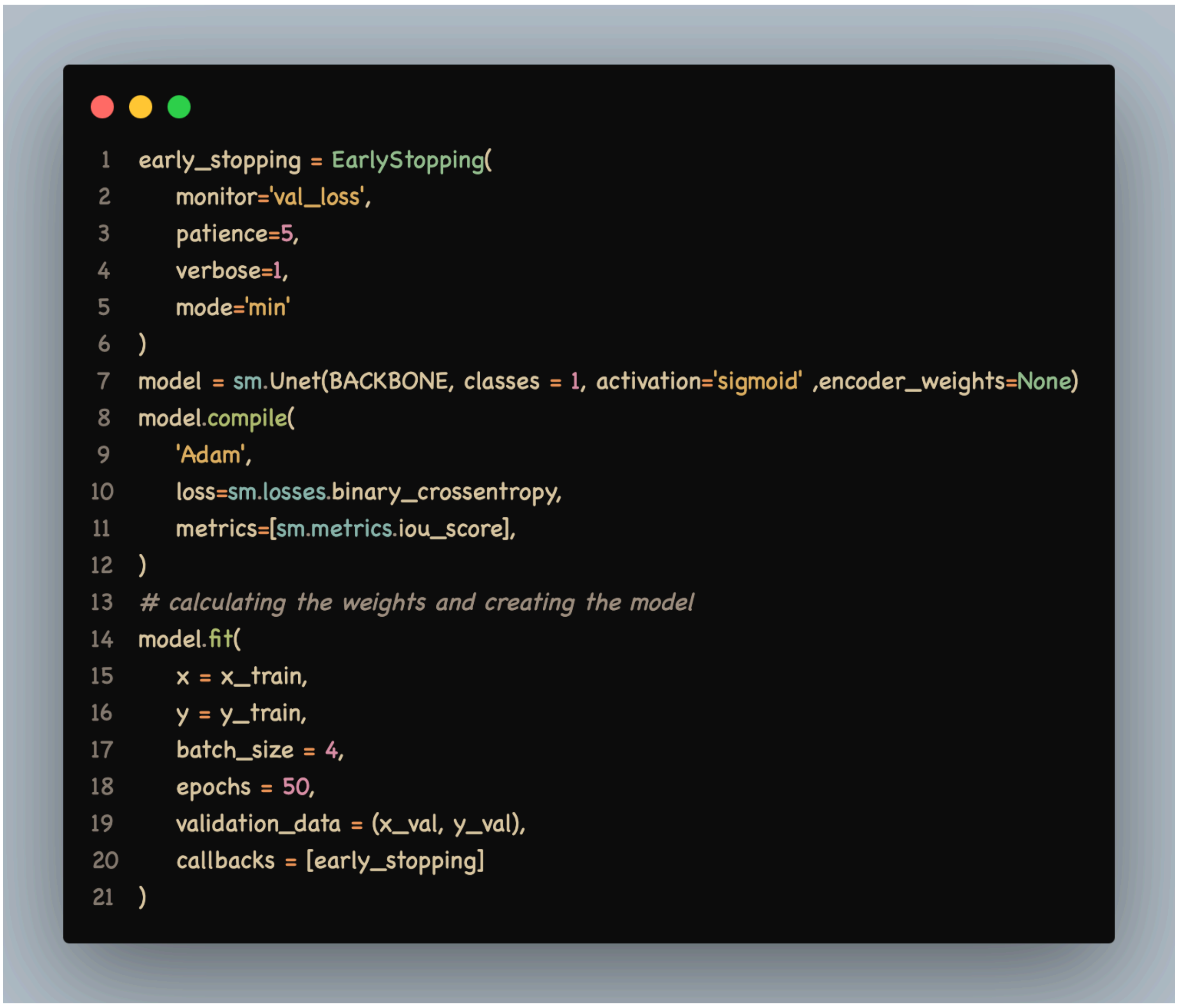
- Further, we split the dataset into training and testing part using the train_test_split function of sklearn.model_selection and with go with a 80-20 split of the data.



```python
preprocess_input = get_preprocessing(BACKBONE)

x= images
y= labels

# converting the pixel value to be in the range of 0 to 1 and of float32 type
x = x.astype(np.float32)/255.0
y = y.astype(np.float32)/255.0

# splitting the dataset into training and testing part, getting a 80-20 split
x_train, x_val, y_train, y_val = train_test_split(x, y , test_size=0.2, random_state=42)
x_train = preprocess_input(x_train)
x_val = preprocess_input(x_val)
```

6. MODEL CREATION

- The model is created using the Unet architecture with resnet18.



```
early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=5,
    verbose=1,
    mode='min'
)
model = sm.Unet(BACKBONE, classes = 1, activation='sigmoid' ,encoder_weights=None)
model.compile(
    'Adam',
    loss=sm.losses.binary_crossentropy,
    metrics=[sm.metrics.iou_score],
)
# calculating the weights and creating the model
model.fit(
    x = x_train,
    y = y_train,
    batch_size = 4,
    epochs = 50,
    validation_data = (x_val, y_val),
    callbacks = [early_stopping]
)
```

- Since we are dealing with a sort of binary segmentation, we will use no of classes as 1. In addition to this, since out pixel value lie between 0 to 1 and we have binary class segmentation, we will use the sigmoid activation function which enable the possibility of using the threshold value to convert the pixel values of out prediction.

- The metric used for calculating the loss is binary cross entropy and the evaluation metric used is IOU (intersection over union score). The Adam optimiser is used to handle the updation of weights of the network.

- After fine tuning our model multiple times, the best outcome was encountered when we did not use pre trained weights in the encoder portion of resnet18 and trained the weights from scratch, thus we use the encoder_weights = None parameter.

7. MODEL FINE TUNING

- In order to fine tune our predictions, different models were build with slight differences.

- Changes in the learning rate of the optimiser, choosing a different backbone for the Unet architecture, changing the number of batches, changing the activation function, using pre-trained encoder weights, change no of epochs etc. steps were taken to get the best model.

- The best predictions were encountered when the parameters where as following:

  - Activation function : sigmoid

  - Optimiser : Adam

  - Loss : binary cross entropy

  - Evaluation metric : IOU score

  - Backbone : resnet18

  - Batch size : 4

  - Epochs : 50

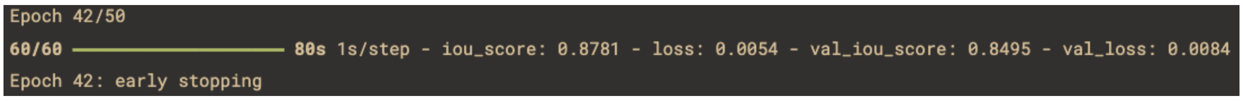
8. MODEL PERFORMANCE

- The performance metrics of the model when fit with the data are :

  - iou_score : 0.8781

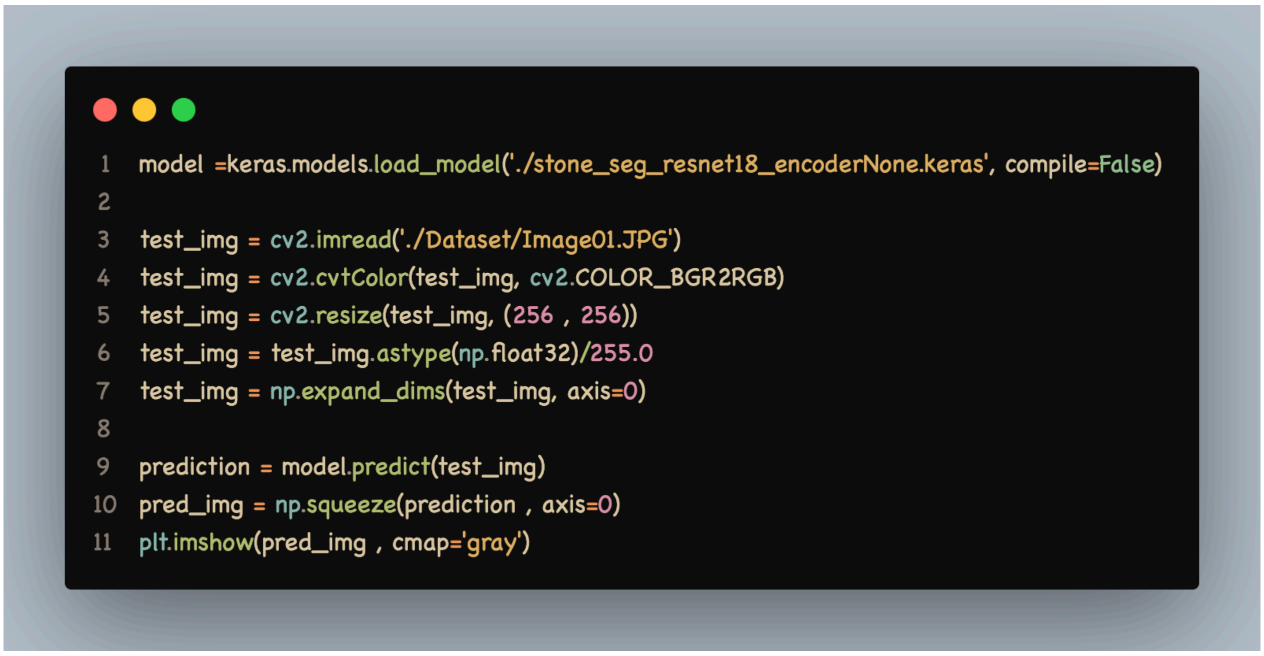
- Loss : 0.0054
- val_iou_score : 0.8495
- val_loss : 0.0084

9. HOW TO PREDICT?



```
Epoch 42/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 80s 1s/step - iou_score: 0.8781 - loss: 0.0054 - val_iou_score: 0.8495 - val_loss: 0.0084
Epoch 42: early stopping
```

- To get the prediction for a test image, first load the image using opencv.
- The image must be resize to the size of the labels used in the model.
- Convert the images pixels to be in the range of 0 to 1 and of float32 type.
- Expand the dimension of the image on axis=0 so that our model understands that there is only one test image.



```python
model =keras.models.load_model('./stone_seg_resnet18_encoderNone.keras', compile=False)

test_img = cv2.imread('./Dataset/Image01.JPG')
test_img = cv2.cvtColor(test_img, cv2.COLOR_BGR2RGB)
test_img = cv2.resize(test_img, (256 , 256))
test_img = test_img.astype(np.float32)/255.0
test_img = np.expand_dims(test_img, axis=0)

prediction = model.predict(test_img)
pred_img = np.squeeze(prediction , axis=0)
plt.imshow(pred_img , cmap='gray')
```