Research Configuration Manual

MSc Research Project
Data Analytics

Nikhil Vaidya
Student ID: x20245980

School of Computing
National College of Ireland

Supervisor: Prof. Qurrat Ul Ain

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Nikhil Vaidya
Student ID:	x20245980
Programme:	Data Analytics
Year:	2021
Module:	MSc Research Project
Supervisor:	Prof. Qurrat Ul Ain
Submission Due Date:	18/09/2022
Project Title:	Research Configuration Manual
Word Count:	988
Page Count:	9

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Nikhil Vaidya
Date:	18/09/2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Research Configuration Manual

Nikhil Vaidya
x20245980

1 Introduction

This configuration manual provides information on the activities and tasks carried out during this project's implementation phase. The hardware and software specifications are provided in case this project needs to be duplicated in the future. The documents detail every stage of code development and deployment. Additionally, it addresses the prerequisites for the code to function as intended.

2 System Configuration

2.1 Hardware Configurations

Below Figure 1 depicts the hardware configuration used in ourt research.

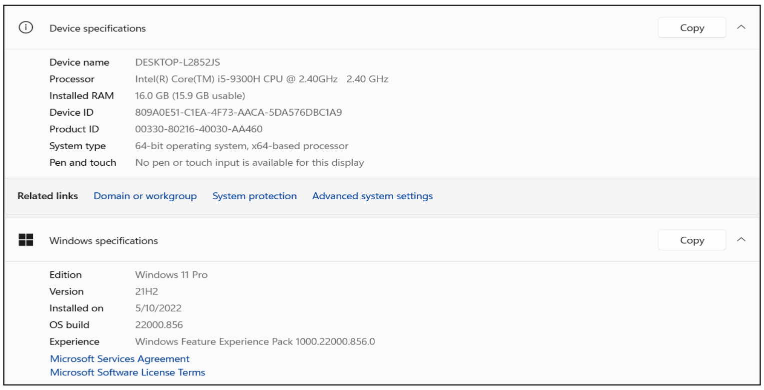


Figure 1: Hardware configurations

2.2 Software Configurations

The software configuration section is the most crucial section of the implementation. The section highlights important software requirements for our research.

2.2.1 Google Colab Pro

Google colab pro version is used for implementing the code in this project. Google colab is an online service that provides free computing resources such as GPU and CPU for running the python code. All the necessary libraries are loaded in the google colab. The dataset is saved on google drive and retrieved using the below-given code:

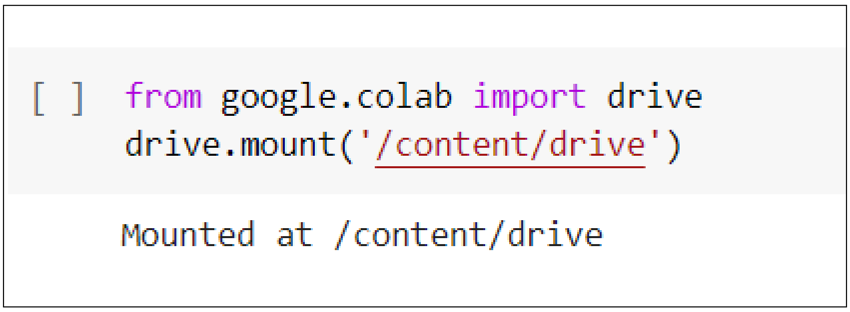


Figure 2: Accessing Google Drive on Colab

After the code is executed, an authorization link is generated, which we have to permit by clicking on the link.Figure 2 shows how the code is implemented.

Using the colab pro version, we can switch to a GPU or TPU runtime for much faster and better resources. This feature can be implemented by changing the "Change Runtime Type" in the runtime menu, as shown in the figure.

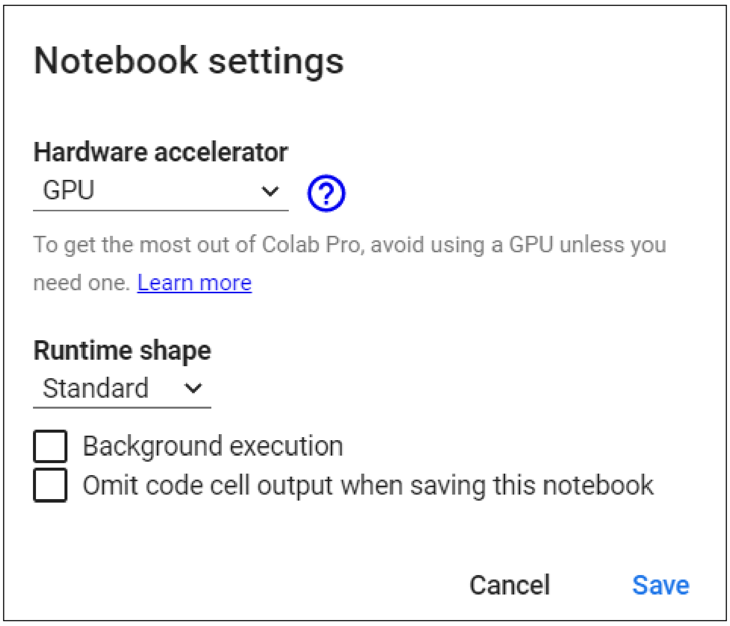


Figure 3: GPU Runtime In Google Colab

2.2.2 Overleaf

Overleaf was used to create the report associated with this research project. Overleaf is an online document creation platform that uses the Latex language to format the document. It is widely used for various report creation and is known for its simplicity and real-time document snapshot display. Below given figure shows the UI of Overleaf.

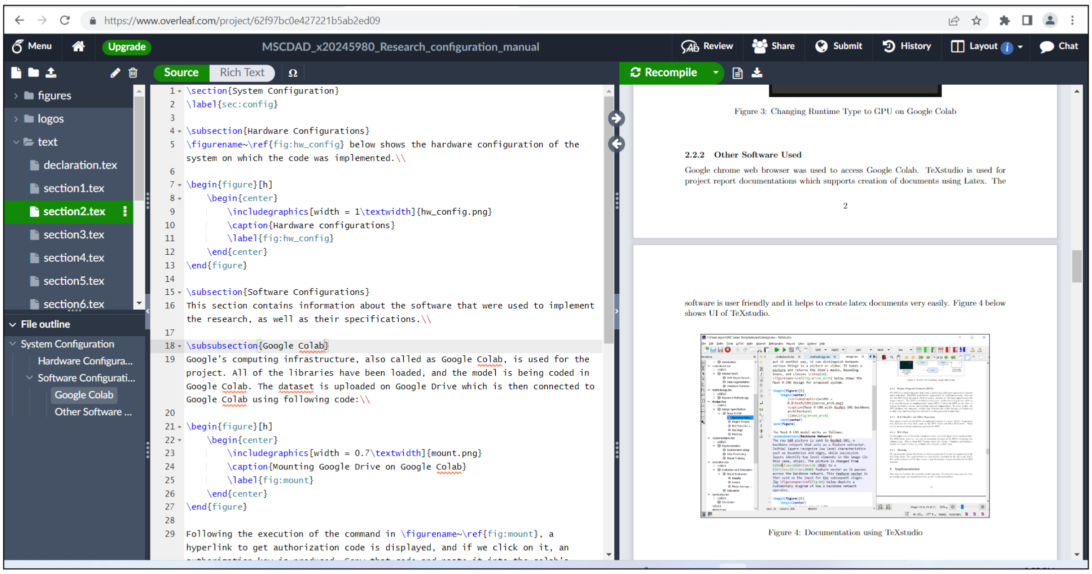


Figure 4: Overleaf

3 Data Preparation

Dataset used in this research is acquried from ISIC (International Skin Imaging Collaboration) lesion segmentation and classification Challenge[1] depicted in Figure 5.

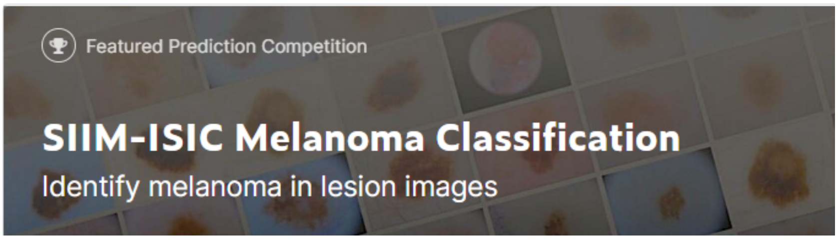


Figure 5: ISIC Skin Lesion Challenge Dataset

Figure 7 how to upload the dataset on the google drive.

[1]https://challenge.isic-archive.com/data/

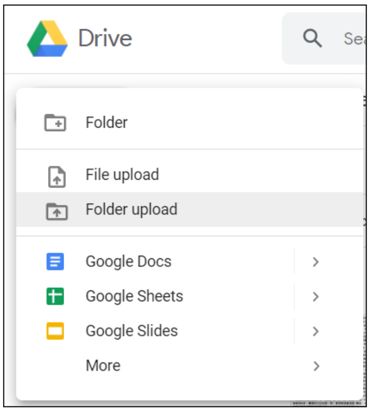


Figure 6: Dataset Upload

The dataset has two folders and two files: train and val, which are further subdivided into four folders. And two ground truth files. After uploading and unzipping the dataset on the drive, the folder structure required on the drive to replicate the code is depicted in Figure 7.

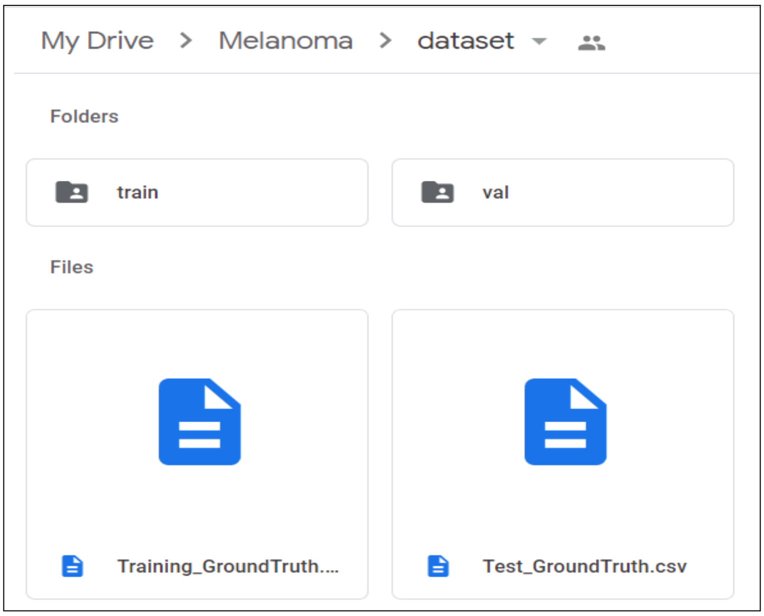


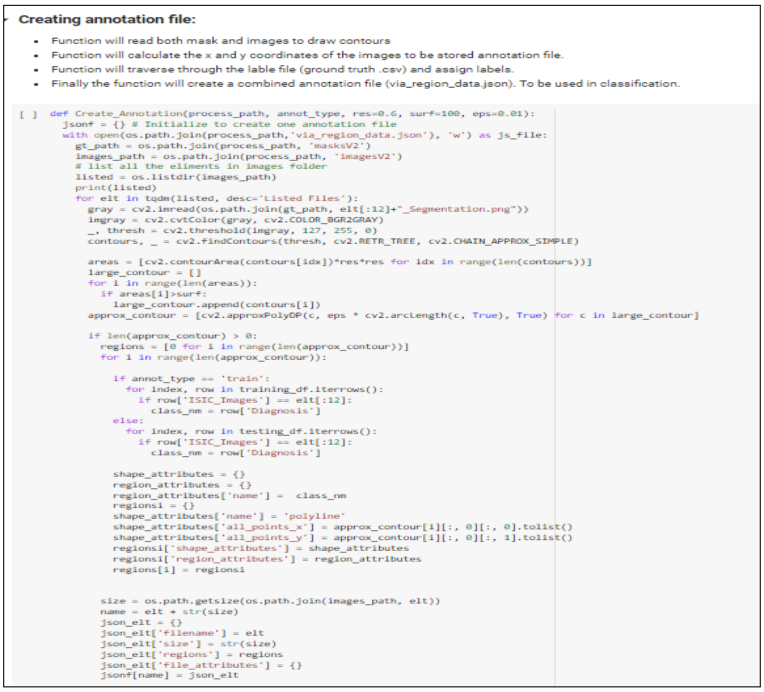
Figure 7: Folder Structure

The train and val folder are further subdivided into four folders each: images and masks, which are original data downloaded from the website. The other two folders are imagesV2 and masksV2, which are resized images of 512x512 resolution. The train and val folder also contains an annotation file via_region_data.json, which we created while running the code and will be replaced when the code is rerun. The Figure 8 depicts the train folder structure, and the same will be followed for the val folder.

A custom build function to automate the image annotation process is used in this research. The Figure 9 depicts how the annotation function is implemented to build the

Figure 8: Train Folder Structure

annotation file in google drive named via_region_data.json.



Creating annotation file:

- Function will read both mask and images to draw contours
- Function will calculate the x and y coordinates of the images to be stored annotation file.
- Function will traverse through the lable file (ground truth .csv) and assign labels.
- Finally the function will create a combined annotation file (via_region_data.json). To be used in classification.

```
[ ] def Create_Annotation(process_path, annot_type, res=0.6, surf=100, eps=0.01):
      jsonf = {} # Initialize to create one annotation file
      with open(os.path.join(process_path,'via_region_data.json'), 'w') as js_file:
        gt_path = os.path.join(process_path, 'masksV2')
        images_path = os.path.join(process_path, 'imagesV2')
        # list all the eliments in images folder
        listed = os.listdir(images_path)
        print(listed)
        for elt in tqdm(listed, desc='Listed Files'):
          gray = cv2.imread(os.path.join(gt_path, elt[:12]+"_Segmentation.png"))
          imgray = cv2.cvtColor(gray, cv2.COLOR_BGR2GRAY)
          _, thresh = cv2.threshold(imgray, 127, 255, 0)
          contours, _ = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

          areas = [cv2.contourArea(contours[idx])*res*res for idx in range(len(contours))]
          large_contour = []
          for i in range(len(areas)):
            if areas[i]>surf:
              large_contour.append(contours[i])
          approx_contour = [cv2.approxPolyDP(c, eps * cv2.arcLength(c, True), True) for c in large_contour]

          if len(approx_contour) > 0:
            regions = [0 for i in range(len(approx_contour))]
            for i in range(len(approx_contour)):

              if annot_type == 'train':
                for index, row in training_df.iterrows():
                  if row['ISIC_Images'] == elt[:12]:
                    class_nm = row['Diagnosis']
              else:
                for index, row in testing_df.iterrows():
                  if row['ISIC_Images'] == elt[:12]:
                    class_nm = row['Diagnosis']

              shape_attributes = {}
              region_attributes = {}
              region_attributes['name'] =  class_nm
              regionsi = {}
              shape_attributes['name'] = 'polyline'
              shape_attributes['all_points_x'] = approx_contour[i][:, 0][:, 0].tolist()
              shape_attributes['all_points_y'] = approx_contour[i][:, 0][:, 1].tolist()
              regionsi['shape_attributes'] = shape_attributes
              regionsi['region_attributes'] = region_attributes
              regions[i] = regionsi


            size = os.path.getsize(os.path.join(images_path, elt))
            name = elt + str(size)
            json_elt = {}
            json_elt['filename'] = elt
            json_elt['size'] = str(size)
            json_elt['regions'] = regions
            json_elt['file_attributes'] = {}
            jsonf[name] = json_elt
```

Figure 9: Annotator Function

4 Model Building

This research implements Mask R-CNN with ResNet101 and ResNet50 backbone using a custom function to create an annotation file, which is the project's novelty. In addition, a transfer learning technique is used to train the model with pre-trained COCO weights. Mask R-CNN is the state-of-the-art approach for image segmentation and classification.(Huang et al.; 2020). We have only described the ResNet101 model implementation in the configuration, and the Resnet50 model will perform the same set of code.

The publicly available Mask R-CNN library is imported in this research from the GitHub link. [2]. We also added our code to implement the customization of our model.

Figure 10 shows how Mask RCNN is imported on google colab.

```
!git clone https://www.github.com/matterport/Mask_RCNN.git

Cloning into 'Mask_RCNN'...
warning: redirecting to https://github.com/matterport/Mask_RCNN.git/
remote: Enumerating objects: 956, done.
remote: Total 956 (delta 0), reused 0 (delta 0), pack-reused 956
Receiving objects: 100% (956/956), 137.67 MiB | 54.30 MiB/s, done.
Resolving deltas: 100% (558/558), done.
```

Figure 10: Importing Mask R-CNN Model

In below given Figure 11), we are copying the data from google drive to the google colab local directory to make the processing even faster.

```
if os.path.exists(ROOT_DIR+'/dataset/'):
  shutil.copytree(DATASET_PATH_DRIVE+'/train/imagesV2/', ROOT_DIR+'/dataset/train/')
  shutil.copytree(DATASET_PATH_DRIVE+'/val/imagesV2/', ROOT_DIR+'/dataset/val/')
  shutil.copy(DATASET_PATH_DRIVE+'/train/via_region_data.json',  ROOT_DIR+'/dataset/train/')
  shutil.copy(DATASET_PATH_DRIVE+'/val/via_region_data.json',  ROOT_DIR+'/dataset/val/')
```

Figure 11: Copying Data to Google Colab Local Repository

Figure 5) shows the LesionISICDataset() class which is used to load the dataset and create train and test data.

```
# Create skin lesion class
class LesionISICDataset(utils.Dataset):
  #Load_lesion90 function to load the dataset
  def load_lesion(self, dataset_dir, subset):
    self.add_class("Lesion", 1, "benign")
    self.add_class("Lesion", 2, "malignant")
    self.class_name_to_ids = {'benign':1,'malignant':2}

    assert subset in ["train", "val"]
    dataset_dir = os.path.join(dataset_dir, subset)
    annotations = json.load(open(os.path.join(dataset_dir, "via_region_data.json")))
    annotations = list(annotations.values())
    annotations = [a for a in annotations if a['regions']]

    for a in annotations:
      if type(a['regions']) is dict:
        polygons = [r['shape_attributes'] for r in a['regions'].values()]
        class_names = [list(r['region_attributes']['name'].keys())[0] for r in a['regions'].values()]
      else:
        polygons = [r['shape_attributes'] for r in a['regions']]
        class_names = [list(r['region_attributes']['name'].keys())[0] for r in a['regions']]

        image_path = os.path.join(dataset_dir, a['filename'])
        image = skimage.io.imread(image_path)
        height, width = image.shape[:2]

        self.add_image(
            "Lesion",
            image_id=a['filename'],
            path=image_path,
            width=width, height=height,
```

Figure 12: Skin Dataset Class

Figure 13) depicts how the trainData and valData are prepared, which will be used in model training.

[2] https://github.com/matterport/Mask_RCNN

```
#Create training dataset
trainData = LesionISICDataset()
trainData.load_lesion(DATASET_PATH,"train")
trainData.prepare()

#Create validation dataset
valData = LesionISICDataset()
valData.load_lesion(DATASET_PATH,"val")
valData.prepare()
```

Figure 13: Prepare Dataset

The below-given figure depicts the augmentation performed on our dataset before implementing it in the model train phase.

```
# Augmentation performed on the lesion images
augmen = iaa.Sequential([
    iaa.OneOf([ ## rotate
        iaa.Affine(rotate=0),
        iaa.Affine(rotate=90),
        iaa.Affine(rotate=180),
        iaa.Affine(rotate=270),
    ]),
    iaa.Fliplr(0.5),
    iaa.Flipud(0.5),
    iaa.OneOf([
        iaa.Multiply((0.9, 1.1)),
        iaa.ContrastNormalization((0.9, 1.1)),
    ]),
    iaa.OneOf([ ## blur or sharpen
        iaa.GaussianBlur(sigma=(0.0, 0.1)),
        iaa.Sharpen(alpha=(0.0, 0.1)),
    ]),
])
```

Figure 14: Augmentation

Figure 15) below depicts the config used in Resnet101. Except for the parameters listed below Rest of the parameters are default present in the mrcnn config.

Resnet101 Backbone Model

```
[ ] class LesionsConfig(Config):
        """Configuration file for training lession dataset and override some default hyperparameters
        """
        # Name for the configuration file
        NAME = "Lesion"
        GPU_COUNT = 1
        IMAGES_PER_GPU = 1

        # Class Number Including back ground
        NUM_CLASSES = 1 + 2  #BG + benign + malignant

        RPN_ANCHOR_SCALES = (8, 16 , 32, 64, 128 )


        # Number steps per epoch, validation step and images dimension
        STEPS_PER_EPOCH = 300
        VALIDATION_STEPS = 150
        IMAGE_MIN_DIM = 512
        IMAGE_MAX_DIM = 512
```

Figure 15: ResNet101 Config

The model is then trained for a total number of 30 epochs with a learning rate decay method. 0.001, 0.0001 and 0.0005. The below-given figure depicts the model training process.

```
## Config Initialization for training
config = LesionsConfig()
config.display()
DEVICE = "/gpu:0"

# Model initiation and loading the coco weights
model_rsnt101_trn = modellib.MaskRCNN(mode="training", config=config, model_dir=LOGS_DIR)
model_rsnt101_trn.load_weights(COCO_MODEL, by_name=True,    exclude=["mrcnn_class_logits", "mrcnn_bbox_fc", "mrcnn_bbox", "mrcnn_mask"])

with tf.device(DEVICE):
  model_rsnt101_trn.train(trainData, valData, epochs=2, layers="heads", learning_rate=config.LEARNING_RATE,augmentation=augmen)
  history = model_rsnt101_trn.keras_model.history.history

  # Run the model with all the layers
  model_rsnt101_trn.train(trainData, valData, epochs=14, layers="all", learning_rate=config.LEARNING_RATE/10,augmentation=augmen)
  new_history = model_rsnt101_trn.keras_model.history.history
  for i in new_history: history[i] = history[i] + new_history[i]
```

Figure 16: Resnet Model Training

After the model is trained, the value of the best epoch is calculated and assigned to the custom weight parameter to run the code in inference mode mentioned in Figure 17).

```
# select trained model
dir_names = next(os.walk(model_rsnt101_trn.model_dir))[1]
key = config.NAME.lower()
dir_names = filter(lambda f: f.startswith(key), dir_names)
dir_names = sorted(dir_names)
fps = []
# Pick last directory
for d in dir_names:
    dir_name = os.path.join(model_rsnt101_trn.model_dir, d)
    # Finding last checkpoint
    checkpoints = next(os.walk(dir_name))[-1]
    checkpoints = filter(lambda f: f.startswith("mask_rcnn"), checkpoints)
    checkpoints = sorted(checkpoints)
    #print(checkpoints)
    if not checkpoints:
        print('No weight files in {}'.format(dir_name))
    else:
        checkpoint = os.path.join(dir_name, checkpoints[best_epoch])
        fps.append(checkpoint)

model_path_cust = sorted(fps)[-1]
print('Weight path for best epoch is: {}'.format(model_path_cust))

custom_WEIGHTS_PATH = model_path_cust
print("Assiging path to custom_WEIGHTS_PATH param.......",custom_WEIGHTS_PATH)
```

Figure 17: Custom Weight

After the custom weights are assigned, the Resnet101 model is again run with the inference mode. Figure 18) below is the code for running the model in inference mode.

```
# creating Inference config
class InferenceConfig(LesionsConfig):
    GPU_COUNT = 1
    IMAGES_PER_GPU = 1

config_inf = InferenceConfig()
config_inf.display()

# Running model in inference mode

model_rsnt101_inf = modellib.MaskRCNN(mode="inference", model_dir=LOGS_DIR,
                              config=config_inf)

# Loading the custom trained weights
print("Loading weights ", custom_WEIGHTS_PATH)
model_rsnt101_inf.load_weights(custom_WEIGHTS_PATH, by_name=True)
```

Figure 18: Inference

After the model is run in the inference mode, the evaluation is done based on three metrics mAP(mean average precision), mAR (mean average recall), and f1-score. The proposed model can achieve a 78.6% mAP score, which suggests that the model is a good fit. Figure 19) depicts the code for the same.

```
%%time
mAP, mAR, F1_score = evaluate_model(trainData, model_rsnt101_inf, InferenceConfig)
print("Training mAP: %.3f" % mAP)
print("Training mAR: %.3f" % mAR)
F1_score = (2 * mAP * mAR)/(mAP + mAR)
print('F1 Score for training data: ', F1_score)

mAP_val, mAR_val, F1_score_val = evaluate_model(valData, model_rsnt101_inf, InferenceConfig)
print("Validation mAP: %.3f" % mAP_val)
print("Validation mAR: %.3f" % mAR_val)
F1_score_val = (2 * mAP_val * mAR_val)/(mAP_val + mAR_val)
print('F1 Score for validation data: ', F1_score_val)
```

Figure 19: Evaluation

Finally, after the evaluation is completed, model prediction is made. Below given figure shows the code snippet for the same. Here we have used the predict() function to predict the classification and the function to visualize the predicted image.

```
#Run detection over same image
print("image id is :",image_id)
image, image_meta, gt_class_id, gt_bbox, gt_mask =\
modellib.load_image_gt(valData, InferenceConfig, image_id)
info = valData.image_info[image_id]

# Run the detection model
results = model_rsnt101_inf.detect([image], verbose=1)

# Displayin the results
x = get_ax(1)
r = results[0]
ax = plt.gca()
visualize.display_instances(image, r['rois'], r['masks'], r['class_ids'], valData.class_names, r['scores'], ax=ax, title="Predictions")
log("gt_class_id", gt_class_id)
log("gt_bbox", gt_bbox)
log("gt_mask", gt_mask)
```

Figure 20: Prediction on Validation Dataset

The notebook and all the other artifacts are provided in the ICT solution's appropriate section.

References

Huang, C., Yu, A., Wang, Y. and He, H. (2020). Skin lesion segmentation based on mask r-cnn, *2020 International Conference on Virtual Reality and Visualization (ICVRV)*, pp. 63–67.