

National Institute of Technology, Srinagar
Department of Information Technology, NIT Srinagar, Hazratbal-190006 Srinagar
Jammu and Kashmir.

DEPARTMENT OF INFORMATION TECHNOLOGY



A REPORT
ON
“ Detection, Classification and Semantic
Segmentation of Apples ”

SUBMITTED BY

Mr. TAJAMUL ASHRAF (2018BITE031)
Mr. MOHAMMAD HASEEB (2018BITE024)
Mr. NAIYER ABBAS (2018BITE056)

UNDER THE GUIDANCE OF

PROF. PABITRA MITRA
(IIT KHARAGPUR)

(Internship Period: October 2020 - March 2021)
(Academic Year: 2020-2021)

ACKNOWLEDGEMENT

It gives us great pleasure and satisfaction in presenting this project report on “Detection, Classification and Semantic Segmentation of Apples”.

We would like to express our deep sense of gratitude towards **Prof. Pabitra Mitra** for providing us this great opportunity to work under his guidance and steering us at every step during this research internship.

*We have furthermore to thank **Mr. Nadeem Yousuf** for encouraging us to go ahead and for his continuous guidance.*

We would like to thank all those, who have directly or indirectly helped us for the completion of the work during this project.

TAJAMUL ASHRAF
MOHAMMAD HASEEB
NAIYER ABBAS

Contents

1	Introduction	1
1.1	Problem Statement	1
1.2	Techniques Used	2
2	Detection and classification of Apples	3
2.1	Step1 — Getting Training Data	3
2.2	Step2 — Training our model	3
2.3	Step 3 — Predicting using our trained Model	5
3	Semantic Segmentation	7
3.1	U-Net	8
3.2	Training	9
3.3	Inference	10
3.4	Our Results on training set	10
4	Results	13
5	Conclusion	14

Chapter 1

Introduction

It was a challenging problem that involves building upon methods for apple recognition (e.g. where are they), object localization (e.g. what are their extent), and object classification (e.g. rotten or fresh in our case). After doing some research, we found that in recent years, deep learning techniques are achieving state-of-the-art results for object detection, such as on standard benchmark datasets and in computer vision competitions. Notable is the “You Only Look Once,” or YOLO, family of Convolutional Neural Networks that achieve near state-of-the-art results with a single end-to-end model that can perform object detection in real-time.

We would like to discuss about one specific task in Computer Vision called as Semantic Segmentation. Even though researchers have come up with numerous ways to solve this problem, we worked on a particular architecture namely UNET, which use a Fully Convolutional Network Model for the task.

In this project, we discovered how to develop a model for object detection, classification, counting and Semantic Segmentation on the given dataset.

<https://arxiv.org/abs/1909.06441>

After completing this project, we are familiar with the following:

- YOLO-based Convolutional Neural Network family of models for object detection and the most recent variation called YOLOv3.
- The best-of-breed open source library implementation of the YOLOv3 for the Keras deep learning library.
- How to use a pre-trained YOLOv3 to perform object localization, detection, classification(rotten / fresh) and counting on new images.
- The goal of semantic image segmentation is to label each pixel of an image with a corresponding class of what is being represented. Because we’re predicting for every pixel in the image, this task is commonly referred to as dense prediction.

1.1 Problem Statement

To use computer vision to aid agricultural robotics.

1. Identification of fruits in a tree

2. Read and understand a fruit detection algorithm using semantic segmentation.
3. Devise an algorithm for counting fruits.
4. Implement the algorithm and run on above data set.
5. Devise an algorithm to identify fresh and rotten fruit..

1.2 Techniques Used

First the data set used was '**MinneApple: A Benchmark Data-set for Apple Detection and Segmentation**'

A new dataset to advance the state-of-the-art in fruit detection, segmentation, and counting in orchard environments. While there has been significant recent interest in solving these problems, the lack of a unified dataset has made it difficult to compare results. It enables direct comparisons by providing a large variety of high-resolution images acquired in orchards, together with human annotations of the fruit on trees. The fruits are labeled using polygonal masks for each object instance to aid in precise object detection, localization, and segmentation. Additionally, the dataset data for patch-based counting of clustered fruits. Our dataset contains over 41, 000 annotated object instances in 1000 images. It presents a detailed overview of the dataset together with baseline performance analysis for bounding box detection, segmentation, and fruit counting as well as representative results for yield estimation.

For Detection and Classification we used the all new **YOLO V3 model**.

YOLO V3 uses a variant of Dark-net, which originally has 53 layer network trained on Imagenet. For the task of detection, 53 more layers are stacked onto it, giving us a 106 layer fully convolutional underlying architecture for YOLO v3.

For Semantic Segmentation of Apples we used **U-NET**

U-Net takes its name from the architecture, which when visualized, appears similar to the letter U, as shown in the figure above. Input images are obtained as a segmented output map. The most special aspect of the architecture in the second half. The network does not have a fully-connected layer. Only the convolution layers are used. Each standard convolution process is activated by a ReLU activation function.

As it's commonly known, the dimension reduction process in the height and width that we apply throughout the convolutional neural network—that is, the pooling layer — is applied in the form of a dimension increase in the second half of the model.

Chapter 2

Detection and classification of Apples

We used Artificial Intelligence through the use of computer vision and deep learning to automate apple detection, sorting and identification of rotten ones.

2.1 Step1 — Getting Training Data

We provided sufficient sample images (dataset) of the apples we need the model to detect and identify. We amended the YOLOV3 algorithm for detecting and counting apples as well as identify rotten apples in images. We used the following dataset:

https://github.com/OlafenwaMoses/AppleDetection/releases/download/v1/apple_detection_dataset.zip

It is divided into:

1. 563 images for training the AI model
2. 150 images for testing the trained AI model

2.2 Step2 — Training our model

To generate our model, we trained a YOLOv3 model using ImageAI.

YOLO V3 is designed to be a multi-scaled detector, we also need features from multiple scales. Therefore, features from last three residual blocks are all used in the later detection. We are assuming the input is 416x416, so three scale vectors would be 52x52, 26x26, and 13x13.

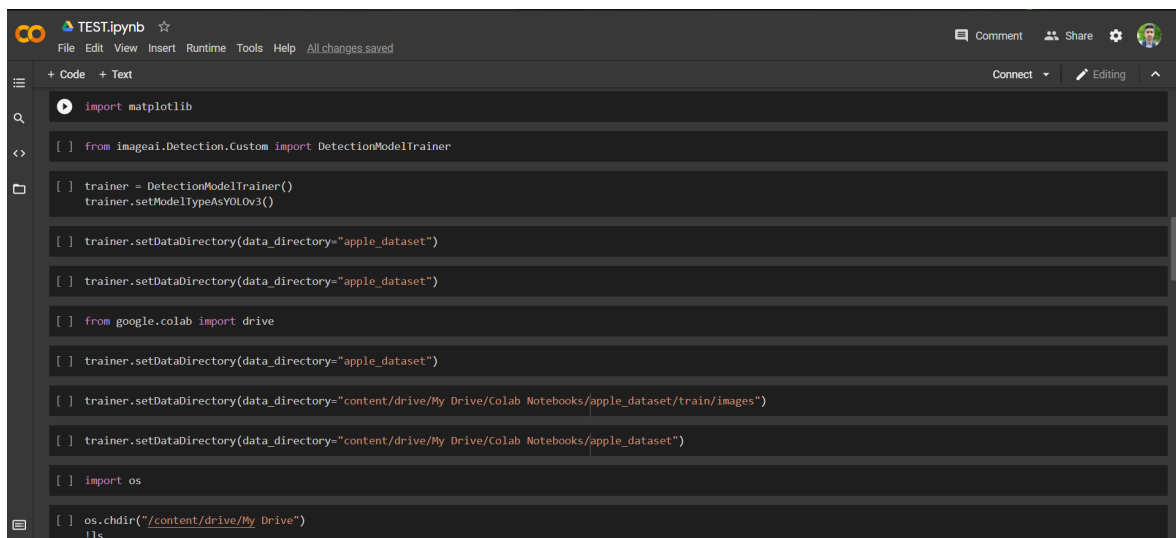
Since the convolution outputs a square matrix of feature values (like 13x13, 26x26, and 52x52 in YOLO), we define this matrix as a “grid” and assign anchor boxes to each cell of the grid. In other words, anchor boxes anchor to the grid cells, and they share the same centroid.

With the final detection output, we can calculate the loss against the ground truth labels now. The loss function consists of four parts (or five, if you split noobj and obj): centroid (xy) loss, width and height (wh) loss, objectness (obj and noobj) loss and classification loss.

```

Loss = Lambda_Coord * Sum(Mean_Square_Error((tx, ty),
(tx', ty') * obj_mask)
+ Lambda_Coord * Sum(Mean_Square_Error((tw, th), (tw',
th') * obj_mask)
+ Sum(Binary_Cross_Entropy(obj, obj') * obj_mask) +
Lambda_Noobj * Sum(Binary_Cross_Entropy(obj, obj') * (1
-obj_mask) * ignore_mask)
+ Sum(Binary_Cross_Entropy(class, class'))
    
```

The below lines of code is to initiate the training on our apple dataset. Now we import the “DetectionModelTrainer” class from ImageAI. After that we created an instance of the class and set our model type to YOLOv3 ,we set the path to our apple dataset and finally we specified the parameters.



```

import matplotlib

[ ] from imageai.Detection.Custom import DetectionModelTrainer

[ ] trainer = DetectionModelTrainer()
    trainer.setModelTypeAsYOLOv3()

[ ] trainer.setDataDirectory(data_directory="apple_dataset")

[ ] trainer.setDataDirectory(data_directory="apple_dataset")

[ ] from google.colab import drive

[ ] trainer.setDataDirectory(data_directory="apple_dataset")

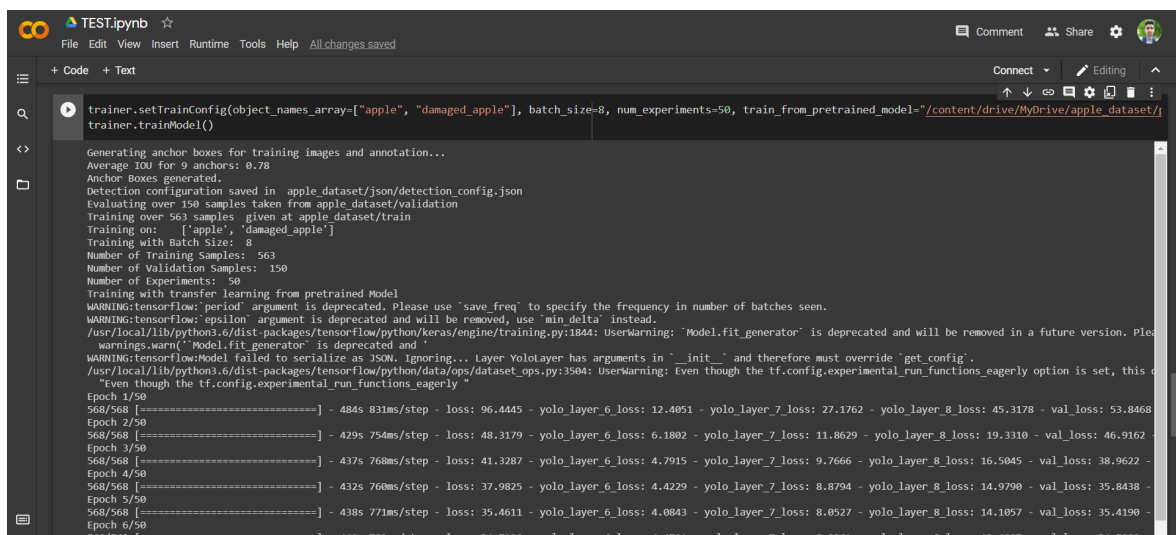
[ ] trainer.setDataDirectory(data_directory="content/drive/My Drive/Colab Notebooks/apple_dataset/train/images")

[ ] trainer.setDataDirectory(data_directory="content/drive/My Drive/Colab Notebooks/apple_dataset")

[ ] import os

[ ] os.chdir("/content/drive/My Drive")
    
```

ImageAI will create apple dataset/models folder which is where all generated models will be saved.

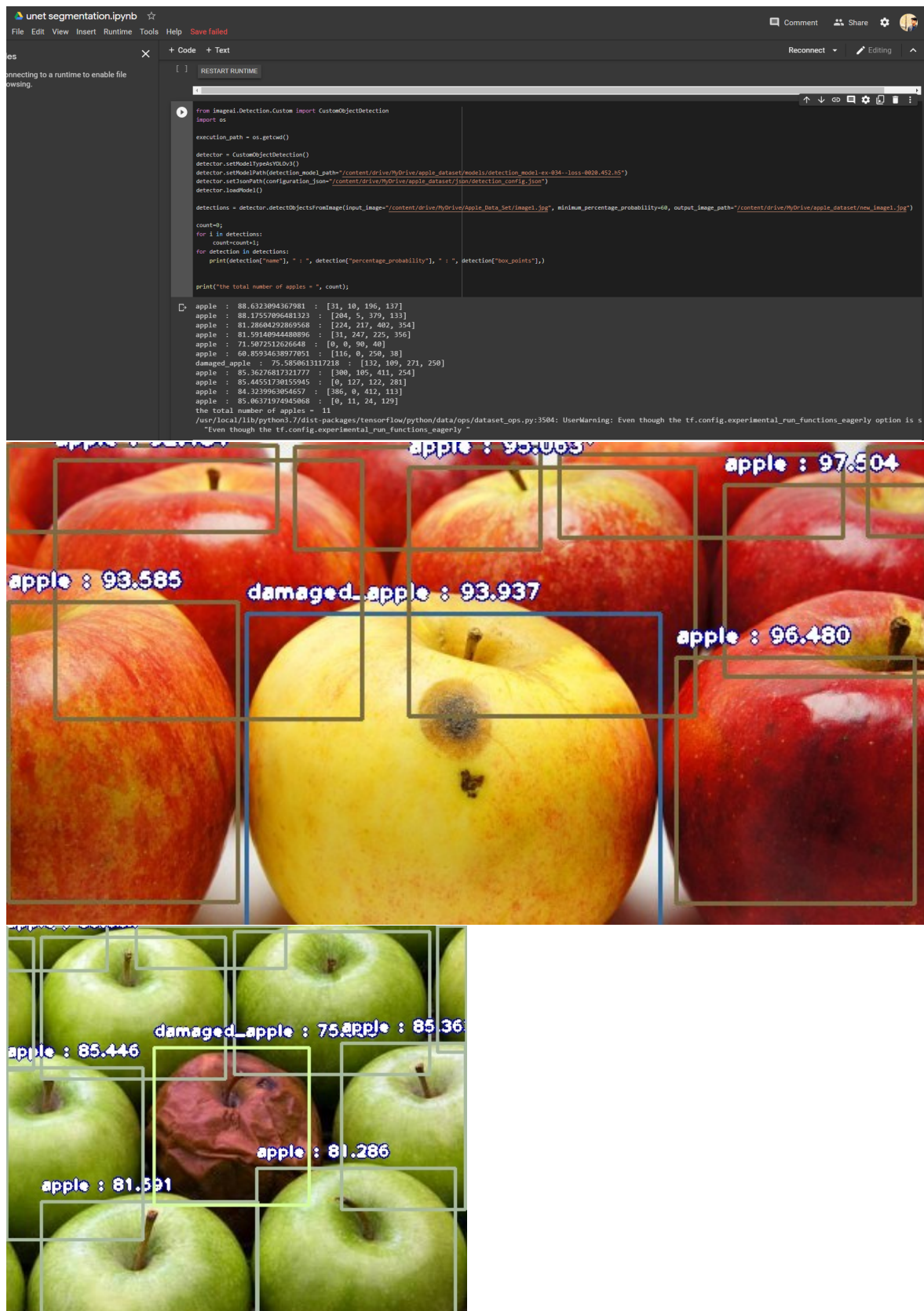


```

trainer.setTrainConfig(object_names_array=["apple", "damaged_apple"], batch_size=8, num_experiments=50, train_from_pretrained_model="/content/drive/MyDrive/apple_dataset/f
trainer.trainModel()

Generating anchor boxes for training images and annotation...
Average IOU for 9 anchors: 0.78
Anchor boxes generated.
Detection configuration saved in apple_dataset/json/detection_config.json
Evaluating over 150 samples taken from apple_dataset/validation
Training over 563 samples given at apple_dataset/train
Training on: ['apple', 'damaged_apple']
Training with Batch Size: 8
Number of Training Samples: 563
Number of Validation Samples: 150
Number of Experiments: 50
Training with transfer learning from pretrained Model
WARNING:tensorflow:period argument is deprecated. Please use 'save_freq' to specify the frequency in number of batches seen.
WARNING:tensorflow:epsilon argument is deprecated and will be removed, use 'min_delta' instead.
/usr/local/lib/python3.6/dist-packages/tensorflow/python/keras/engine/training.py:1844: UserWarning: 'Model.fit_generator' is deprecated and will be removed in a future version. Please use 'Model.fit' instead.
warnings.warn('Model.fit_generator' is deprecated and will be removed in a future version. Please use 'Model.fit' instead.')
WARNING:tensorflow:Model failed to serialize as JSON. Ignoring... Layer YoloLayer has arguments in '__init__' and therefore must override 'get_config'.
/usr/local/lib/python3.6/dist-packages/tensorflow/python/data/ops/dataset_ops.py:3504: UserWarning: Even though the tf.config.experimental_run_functions_eagerly option is set, this operation will not be run eagerly.
warnings.warn('Even though the tf.config.experimental_run_functions_eagerly option is set, this operation will not be run eagerly.')
Epoch 1/50
568/568 [=====] - 484s 831ms/step - loss: 96.4445 - yolo_layer_6_loss: 12.4051 - yolo_layer_7_loss: 27.1762 - yolo_layer_8_loss: 45.3178 - val_loss: 53.8468
Epoch 2/50
568/568 [=====] - 429s 754ms/step - loss: 48.3179 - yolo_layer_6_loss: 6.1802 - yolo_layer_7_loss: 11.8629 - yolo_layer_8_loss: 19.3310 - val_loss: 46.9162
Epoch 3/50
568/568 [=====] - 437s 768ms/step - loss: 41.3287 - yolo_layer_6_loss: 4.7915 - yolo_layer_7_loss: 9.7666 - yolo_layer_8_loss: 16.5045 - val_loss: 38.9622
Epoch 4/50
568/568 [=====] - 432s 760ms/step - loss: 37.9825 - yolo_layer_6_loss: 4.4229 - yolo_layer_7_loss: 8.8794 - yolo_layer_8_loss: 14.9790 - val_loss: 35.8438
Epoch 5/50
568/568 [=====] - 438s 771ms/step - loss: 35.4611 - yolo_layer_6_loss: 4.0843 - yolo_layer_7_loss: 8.0527 - yolo_layer_8_loss: 14.1057 - val_loss: 35.4190
Epoch 6/50
568/568 [=====] - 440s 780ms/step - loss: 34.7196 - yolo_layer_6_loss: 4.4701 - yolo_layer_7_loss: 8.0361 - yolo_layer_8_loss: 13.4007 - val_loss: 34.5099
    
```

2.3 Step 3 — Predicting using our trained Model



As we can see, the trained apple detection model is able to detect all the apples in the image as well as identify the apple with a defect (apple with a spot). You can use this trained model to count apples

```
count=0;
for i in detections:
    count=count+1;
for detection in detections:
    print(detection["name"], " : ", detection["percentage_probability"], " : ", detection["box_points"],)

print("the total number of apples = ", count);

apple : 88.6323094367981 : [31, 10, 196, 137]
apple : 88.17557096481323 : [204, 5, 379, 133]
apple : 81.28604292869568 : [224, 217, 402, 354]
apple : 81.59140944480896 : [31, 247, 225, 356]
apple : 71.5072512626648 : [0, 0, 90, 40]
apple : 60.85934638977051 : [116, 0, 250, 38]
damaged_apple : 75.5850613117218 : [132, 109, 271, 250]
apple : 85.36276817321777 : [300, 105, 411, 254]
apple : 85.44551730155945 : [0, 127, 122, 281]
apple : 84.3239963054657 : [386, 0, 412, 113]
apple : 85.06371974945068 : [0, 11, 24, 129]
the total number of apples = 11
```

The complete code along with other predicted images can be accessed here:

https://github.com/Tajamul21/Detection-Classification-and-Semantic_Segmentation-of-apples

Chapter 3

Semantic Segmentation

The goal of semantic image segmentation is to label each pixel of an image with a corresponding class of what is being represented. Because we're predicting for every pixel in the image, this task is commonly referred to as dense prediction.

Unlike the previous tasks, the expected output in semantic segmentation are not just labels and bounding box parameters. The output itself is a high resolution image (typically of the same size as input image) in which each pixel is classified to a particular class. Thus it is a pixel level image classification.

The different operations that are typically used in a Convolutional Network. Please make a note of the terminologies used.

i. Convolution operation

There are two inputs to a convolutional operation i) A 3D volume (input image) of size ($n_{in} \times n_{in} \times \text{channels}$)

ii) A set of 'k' filters (also called as kernels or feature extractors) each one of size ($f \times f \times \text{channels}$), where f is typically 3 or 5.

The output of a convolutional operation is also a 3D volume (also called as output image or feature map) of size ($n_{out} \times n_{out} \times k$).

The relationship between n_{in} and n_{out} is as follows:

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$

n_{in} : number of input features

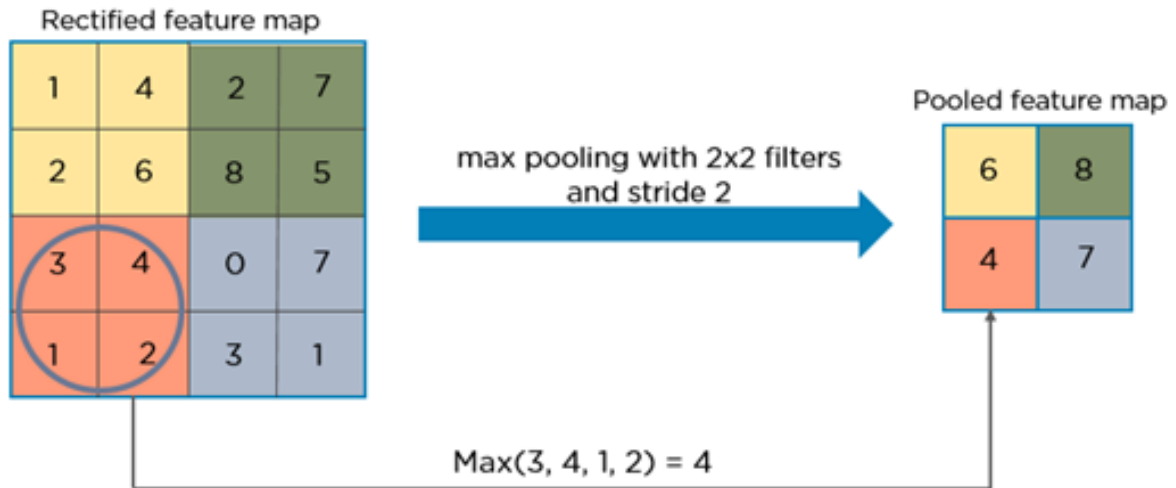
n_{out} : number of output features

k : convolution kernel size

p : convolution padding size

s : convolution stride size

The function of pooling is to reduce the size of the feature map so that we have fewer parameters in the network. the size of the filter and strides are two important hyper-parameters in the max pooling operation.



3.1 U-Net

U-Net is an architecture for semantic segmentation. It consists of a contracting path and an expansive path. The contracting path follows the typical architecture of a convolutional network. It consists of the repeated application of two 3x3 convolutions (unpadded convolutions), each followed by a rectified linear unit (ReLU) and a 2x2 max pooling operation with stride 2 for downsampling. At each downsampling step we double the number of feature channels. Every step in the expansive path consists of an upsampling of the feature map followed by a 2x2 convolution (“up-convolution”) that halves the number of feature channels, a concatenation with the correspondingly cropped feature map from the contracting path, and two 3x3 convolutions, each followed by a ReLU. The cropping is necessary due to the loss of border pixels in every convolution. At the final layer a 1x1 convolution is used to map each 64-component feature vector to the desired number of classes. In total the network has 23 convolutional layers.

```

def get_unet(input_img, n_filters = 16, dropout = 0.1, batchnorm = True):
    """Function to define the UNET Model"""
    # Contracting Path
    c1 = conv2d_block(input_img, n_filters * 1, kernel_size = 3, batchnorm = batchnorm)
    p1 = MaxPooling2D((2, 2))(c1)
    p1 = Dropout(dropout)(p1)

    c2 = conv2d_block(p1, n_filters * 2, kernel_size = 3, batchnorm = batchnorm)
    p2 = MaxPooling2D((2, 2))(c2)
    p2 = Dropout(dropout)(p2)

    c3 = conv2d_block(p2, n_filters * 4, kernel_size = 3, batchnorm = batchnorm)
    p3 = MaxPooling2D((2, 2))(c3)
    p3 = Dropout(dropout)(p3)

    c4 = conv2d_block(p3, n_filters * 8, kernel_size = 3, batchnorm = batchnorm)
    p4 = MaxPooling2D((2, 2))(c4)
    p4 = Dropout(dropout)(p4)

    c5 = conv2d_block(p4, n_filters * 16, kernel_size = 3, batchnorm = batchnorm)

    # Expansive Path
    u6 = Conv2DTranspose(n_filters * 8, (3, 3), strides = (2, 2), padding = 'same')(c5)
    u6 = concatenate([u6, c4])
    u6 = Dropout(dropout)(u6)
    c6 = conv2d_block(u6, n_filters * 8, kernel_size = 3, batchnorm = batchnorm)

    u7 = Conv2DTranspose(n_filters * 4, (3, 3), strides = (2, 2), padding = 'same')(c6)
    u7 = concatenate([u7, c3])
    u7 = Dropout(dropout)(u7)
    c7 = conv2d_block(u7, n_filters * 4, kernel_size = 3, batchnorm = batchnorm)

    u8 = Conv2DTranspose(n_filters * 2, (3, 3), strides = (2, 2), padding = 'same')(c7)
    u8 = concatenate([u8, c2])
    u8 = Dropout(dropout)(u8)
    c8 = conv2d_block(u8, n_filters * 2, kernel_size = 3, batchnorm = batchnorm)

    u9 = Conv2DTranspose(n_filters * 1, (3, 3), strides = (2, 2), padding = 'same')(c8)
    u9 = concatenate([u9, c1])
    u9 = Dropout(dropout)(u9)
    c9 = conv2d_block(u9, n_filters * 1, kernel_size = 3, batchnorm = batchnorm)

    outputs = Conv2D(1, (1, 1), activation='sigmoid')(c9)
    model = Model(inputs=[input_img], outputs=[outputs])
    return model

```

3.2 Training

Model is compiled with Adam optimizer and we use binary cross entropy loss function since there are only two classes (apple and no apple). We use Keras callbacks to implement:

Learning rate decay if the validation loss does not improve for 5 continues epochs.

Early stopping if the validation loss does not improve for 10 continues epochs.

Save the weights only if there is improvement in validation loss.

We use a batch size of 32.

```

[14]: model.summary()

```

Layer (type)	Output Shape	Param #	Connected to
img (InputLayer)	[(None, 128, 128, 1) 0		
conv2d_1 (Conv2D)	(None, 128, 128, 16) 160		img[0][0]
batch_normalization_1 (BatchNor	(None, 128, 128, 16) 64		conv2d_1[0][0]
activation_1 (Activation)	(None, 128, 128, 16) 0		batch_normalization_1[0][0]
max_pooling2d (MaxPooling2D)	(None, 64, 64, 16) 0		activation_1[0][0]
dropout (Dropout)	(None, 64, 64, 16) 0		max_pooling2d[0][0]
conv2d_3 (Conv2D)	(None, 64, 64, 32) 4640		dropout[0][0]
batch_normalization_3 (BatchNor	(None, 64, 64, 32) 128		conv2d_3[0][0]
activation_3 (Activation)	(None, 64, 64, 32) 0		batch_normalization_3[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 32, 32, 32) 0		activation_3[0][0]
dropout_1 (Dropout)	(None, 32, 32, 32) 0		max_pooling2d_1[0][0]
conv2d_5 (Conv2D)	(None, 32, 32, 64) 18496		dropout_1[0][0]
batch_normalization_5 (BatchNor	(None, 32, 32, 64) 256		conv2d_5[0][0]
activation_5 (Activation)	(None, 32, 32, 64) 0		batch_normalization_5[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 16, 16, 64) 0		activation_5[0][0]
dropout_2 (Dropout)	(None, 16, 16, 64) 0		max_pooling2d_2[0][0]
conv2d_7 (Conv2D)	(None, 16, 16, 128) 73856		dropout_2[0][0]
batch_normalization_7 (BatchNor	(None, 16, 16, 128) 512		conv2d_7[0][0]

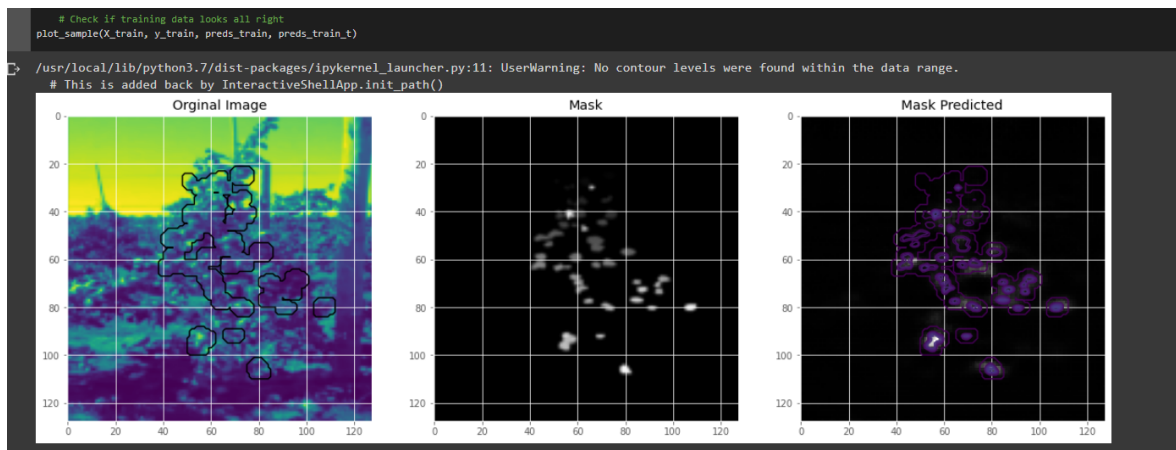
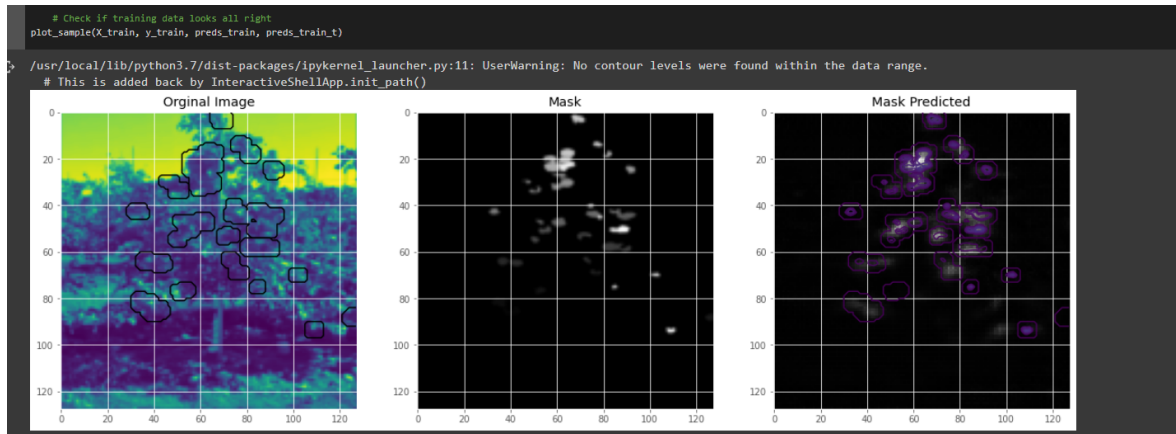
3.3 Inference

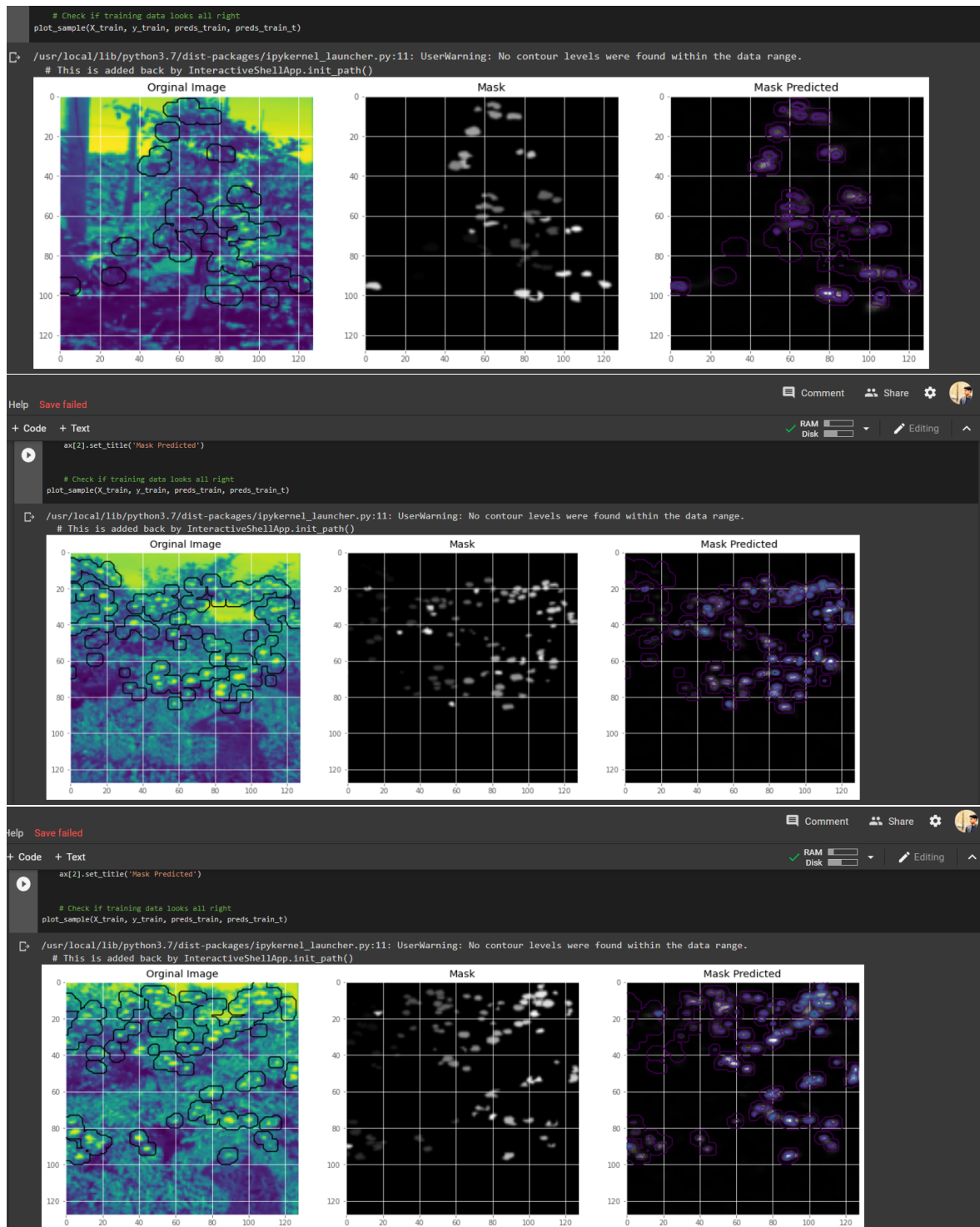
For each pixel we get a value between 0 to 1.

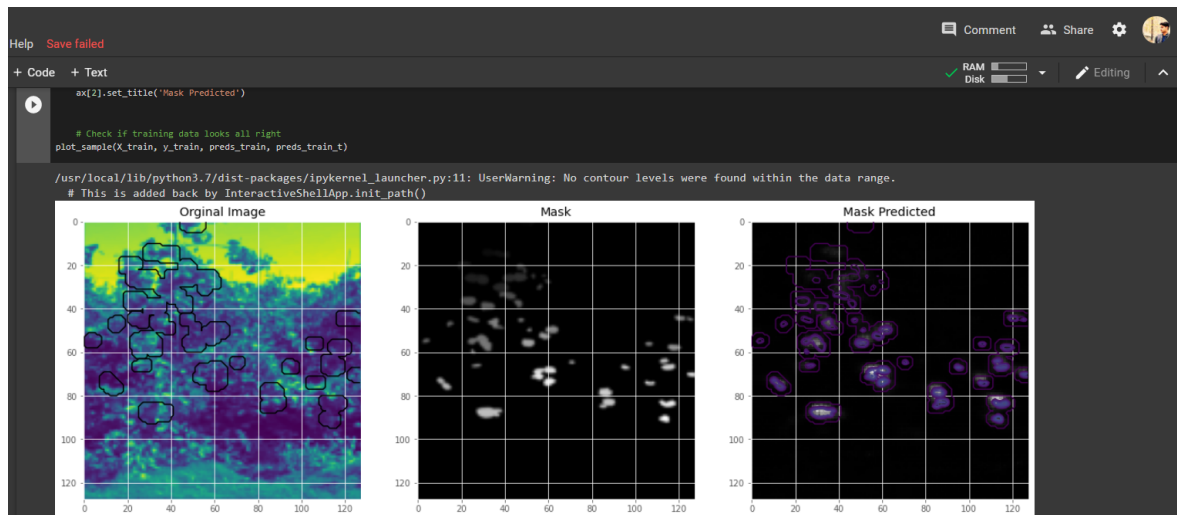
0 represents no apple and 1 represents apple.

We take 0.5 as the threshold to decide whether to classify a pixel as 0 or 1.

3.4 Our Results on training set







The complete code along with other predicted(segmented) images can be accessed here:

https://github.com/Tajamul21/Detection-Classification-and-Semantic_Segmentation-of-apples

Chapter 4

Results

As this Project was divided into two halves, first the detection and classification, the result consisted of all the apples detected in the image and the apple was covered by bounding boxes and anchor boxes.

568/568[=====]-458s804ms/step-loss :
20.8070 - yolo_layer_6_loss : 1.8277 - yolo_layer_7_loss : 4.6716 - yolo_layer_8_loss :
8.2286 - val_loss : 24.7680 - val_yolo_layer_6_loss : 2.1950 - val_yolo_layer_7_loss :
6.1346 - val_yolo_layer_8_loss : 10.3594 Generating anchor boxes for training images and annotation...
Average IOU for 9 anchors : 0.78

Anchor Boxes generated.

Detection configurations saved in apple_dataset/json/detection_config.json

Evaluating over 150 samples taken from apple_dataset/validation

Training over 563 samples given at apple_dataset/train

Training on : ['apple', 'damaged_apple']

Training with Batch Size : 8

Number of Training Samples : 563

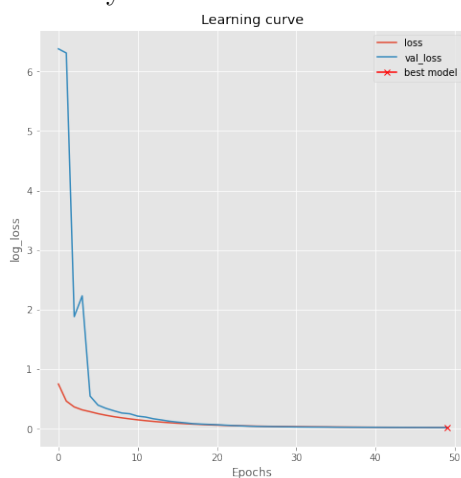
Number of Validation Samples : 150

Number of Experiments : 50

Training with transfer learning from pretrained Model

Now the semantic segmentation of images:

The result was the comparison between the actual mask and the predicted mask. The accuracy achieved in this was about 91.8%.



Chapter 5

Conclusion

We started by studying about the basics of machine learning and its various concepts. It took us almost a month as we were new to this domain. We learnt concepts like regression, gradient descent, and cost functions.

Then we studied about deep learning and computer vision. We studied about Convolutional Neural Networks(CNN), Region Based Convolutional Neural Networks (R-CNN), YOLOV3 model, Activation functions, SVM'S, Back Propagation in deep learning.

In computer vision we studied about Object detection, classification, Image Segmentation and its types. We also learned about transfer learning, U NET and Fine Tuning. We covered all these topics by the month of December. The project work was assigned to us on the 1st of January.

We had to frame a model that should be able to do Detection, Classification, counting and Semantic Segmentation of apples using Tensorflow and Keras.

For detection, classification and counting we amended the pretrained YOLO-V3 model using transfer learning and trained it on our own dataset of fresh and rotten apples. The accuracy we got was around 88%.

Now we were only left with Semantic Segmentation. For this purpose we implemented the U-NET model.

We used the following relationship:

$$Input(128x128x1) \Rightarrow Encoder \Rightarrow (8x8x256) \Rightarrow Decoder \Rightarrow Output(128x128x1)$$

Thus we trained the model using our own dataset(which contained images of apples and their corresponding masks). The model was successful and was able to generate the Semantic Segmentation masks with an accuracy of around 9%.

With this we concluded our project work till the month of March.

After Semantic Segmentation we can modify the above model that would be able to do instance segmentation also. In future we can also extend this model to work in real time using cameras or in pre-recorded videos.

Bibliography

- [1] *MinneApple: A Benchmark Dataset for Apple Detection and Segmentation* January 2020 IEEE Robotics and Automation Letters PP(99):1-1 DOI: 10.1109/LRA.2020.2965061
- [2] *S. W. Chen et al. "Counting apples and oranges with deep learning: A data-driven approach" IEEE Robot. Autom. Lett. vol. 2 no. 2 pp. 781-788 Apr. 2017.*
- [3] *M. Cordts et al. "The Cityscapes Dataset for Semantic Urban Scene Understanding" Proc. IEEE Conf. Comput. Vision Pattern Recognit. pp. 3213-3223 Jun. 2016.*
- [4] *X. Liu et al. Robust fruit counting: Combining deep learning tracking and structure from motion" Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. pp. 1045-1052 Oct. 2018.*
- [5] *O. Ronneberger P. Fischer and T. Brox "U-net: Convolutional networks for biomedical image segmentation" in Proc. Int. Med. image Comput. Comput.-Assisted Intervention Cham Switzerland:Springer pp. 234-241 2015.*
- [6] *M. Rahnemoonfar and C. Sheppard "Deep count: Fruit counting based on deep simulated learning" Sensors vol. 17 no. 4 pp. 905 Apr. 2017.*
- [7] *K. He X. Zhang S. Ren and J. Sun "Deep residual learning for image recognition" Proc. IEEE Conf. Comput. Vision Pattern Recognit. pp. 770-778 Jun. 2016*