HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY AND EDUCATION

FACULTY FOR HIGH QUALITY TRAINING

FINAL PROJECT

ARTIFICIAL INTELLIGENCE

IDENTIFY AND COUNT THE NUMBER OF COWS ON FARMS USING REGION-BASED CONVOLUTION NEURAL NETWORK

STUDENT: TRAN QUOC TOAN – 19146090

ADVISOR: ASSOC. PROF, PH.D.

NGUYEN TRUONG THINH

COURSE: ARIN337629E

Ho Chi Minh City, 20th of June 2022

TABLE OF CONTENTS

ABSTRACT

Object detection in Artificial Intelligence have become so widespread in daily life and industries that they were designed to assist humans in completing tasks. By using object localization algorithm of R-CNN model to identify and count the number of cows on farms, thereby we can easily monitoring the behavior, psychology and health of the cows anytime, from which to plan on motivate cows on going for activities, improves their mood, to produces high quality meat and milk products.

This following project will solve the problem of detect and count number of cows on farms by using Mask R-CNN, one of many object detection algorithm.

CHAPTER 1: INTRODUCTION

1.1. MOTIVATION FOR THE RESEARCH

People, machines, and robots are all required in every factory to produce goods. Humans are in charge of everything, from the smallest to the largest objects, such as computers, robots, and machines. Robots and other electric equipment collaborate to do tasks assigned by humans, assisting the livestock industry's growth.

When appropriate, cutting-edge technologies are used in the livestock industry, the quantity and quality of products increase significantly and the industry continues to grow. In the cattle industry generally, keeping an eye on the animals' health can provide timely, accurate analysis to ensure that the cows are not afflicted with infectious diseases, anorexia, inactivity, or other conditions.

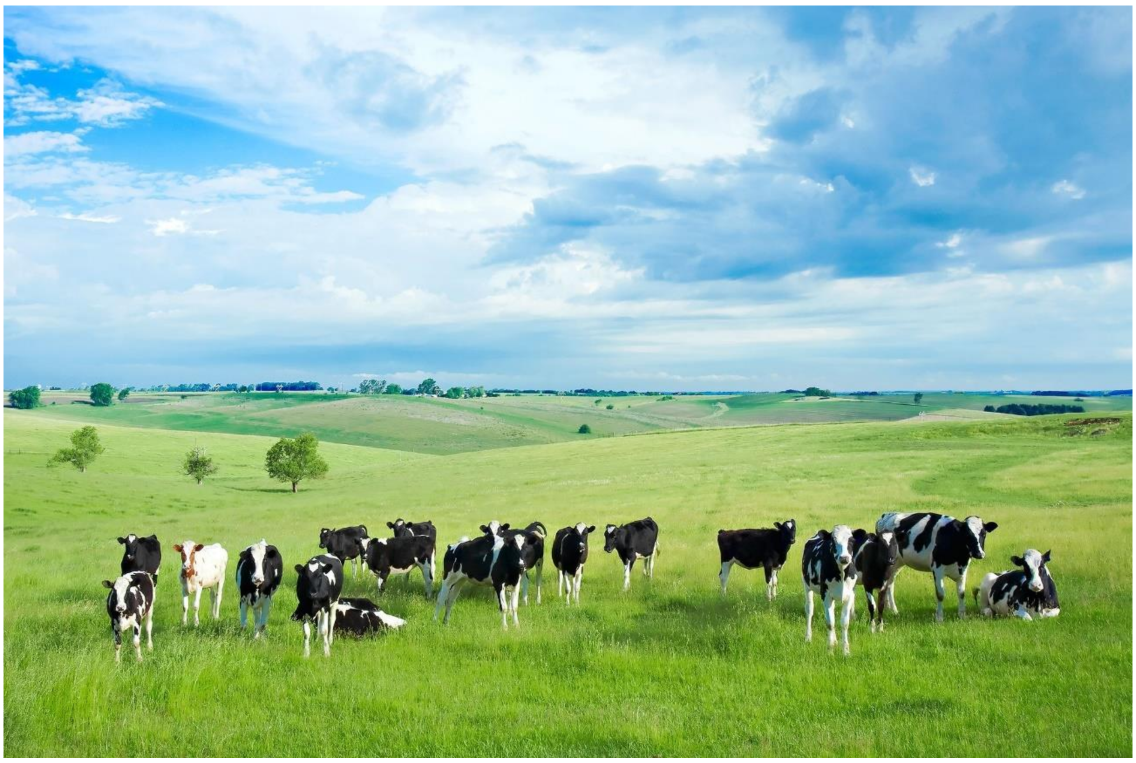


Figure 1.1 Cows in carbon-neutral dairy farm

When hiring people to do the same task over and over, we run into a lot of issues. To create a robotic arm that grabs and places packaged food from the conveyer to the crate in this project. The following are some of the drawbacks of hiring employees:

Employees' health: maybe the most common reason for taking a day off. Even if they try to go to work, they are unable to function at their best while they are sick.

Emotions: many people claim that they will not let their emotions affect their work and that they can handle personal matters without making mistakes.

Conception: despite their differing perspectives, there would be no disagreements or misunderstandings if everyone could fully comprehend each other. The more individuals there are, the more difficult it is for the leaders to communicate their ideas to the employees.

Malpractice: misconduct, running a business is never simple, and not everyone knows what is best for a company. Perhaps they simply believe that if they think for themselves, they can do awful things and profit from it.

There are far more issues with humans than we can list. This is when the AI arrive to solve the issues. AI increase productivity because they do not require rest and follow all of your directions. The opportunity to invest in the surplus value is enormous. They would work continuously, precisely, and quickly for a longer amount of time than humans if the current was not cut off. Compared to today's camera surveillance technology, manual monitoring with the naked eye is very expensive, prone to mistakes, and ineffective.

1.2. PURPOSE

- Study on object detection algorithms
- Capable of building an intelligent monitoring system in livestock industry
- Knowledge of machine learning and deep learning

1.3. RESEARCH METHOD

Object detection algorithms developed from the Region-Based Convolution Neural Network currently typically include Fast R-CNN, Faster R-CNN, Mask R-CNN and Yolo.

Semantic Segmentation Mask R-CNN was designed to be used in this project, aside with Google Colab.

CHAPTER 2: LITERATURE REVIEW

2.1. CONVOLUTIONAL NEURAL NETWORKS

An artificial neural network called a Convolutional Neural Network (CNN) is designed specifically to process pixel data and is used for image recognition and processing. Therefore, the fundamental and fundamental building blocks for the computer vision task of Image Segmentation are Convolutional Neural Networks (CNN Segmentation).

There are three main layers in the convolutional neural network architecture:

- Convolutional layer: Using filters and kernels, this layer aids in abstracting the input image as a feature map.

- Pooling layer: By summarizing the presence of features in different feature map patches, this layer aids in feature map downsampling.

- Fully connected layer: Every neuron in one layer communicates with every other layer's neuron through fully connected layers.

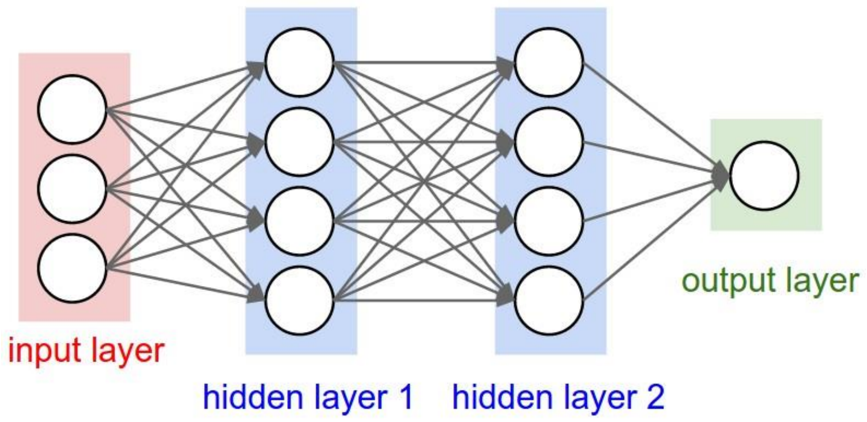


Figure 2.1 Multi layer neural network.

The CNN's built neural network can learn to detect and recognize the object of interest in a picture by combining its layers. For image categorization and object detection using just one object in the image, simple convolutional neural networks are constructed. In a more complex situation with multiple objects in an image, a simple CNN isn't optimal.

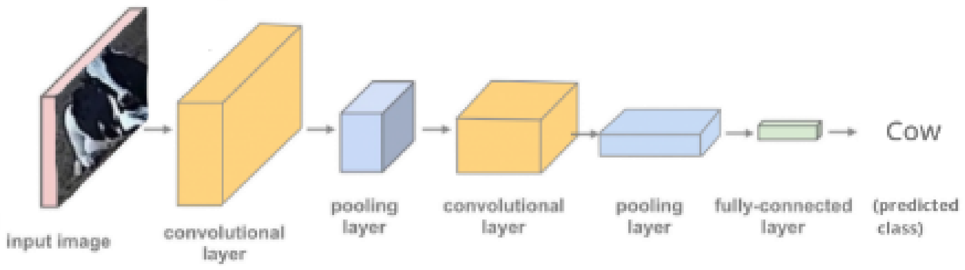


Figure 2.2 Convolutional Neural Network working principle

2.2. REGION-BASED CONVOLUTIONAL NEURAL NETWORK (R-CNN)

2.2.1. Selective search algorithm:

A region suggestion approach for object detection is called Selective Search. It is made to be quick and have a high recall rate. It is based on computing hierarchical grouping of related regions according to compatibility of color, texture, size, and form.

Using a graph-based segmentation method developed by Felzenszwalb and Huttenlocher, Selective Search begins by over-segmenting the image based on pixel intensity. Below is a display of the algorithm's output. Segmented zones are shown in solid color in the image to the right.

Selective Search algorithm takes these oversegments as initial input and performs the following steps [1] :

- Step 1: Add all bounding boxes corresponding to segmented parts to the list of regional proposals.

- Step 2: Group adjacent segments based on similarity

- Step 3: Go to step 1

At each iteration, larger segments are formed and added to the list of region proposals. Hence we create region proposals from smaller segments to larger segments in a bottom-up approach. This is what we mean by computing "hierarchical" segmentations using Felzenszwalb and Huttenlocher's oversegments.

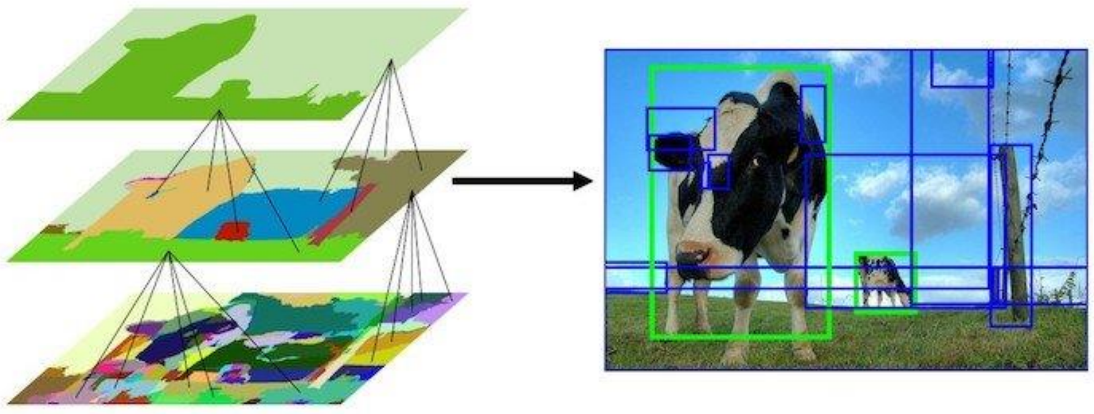


Figure 2.3. Three steps of hierarchical segmentation process

And by that, the R-CNN get about 2000 bounding boxes in the input that are likely to contain objects using the Selective Search algorithm and determines which object is in each bounding box.

2.2.2. Classification of region proposal:

There are many region proposals that contain no objects as a result of the selective search algorithm for up to 2000 region proposals. Therefore, we must add a background layer (which contains no objects). For instance, we will categorize each bounding box in the image below as either a background, a horse, or one of the four region proposals [2].

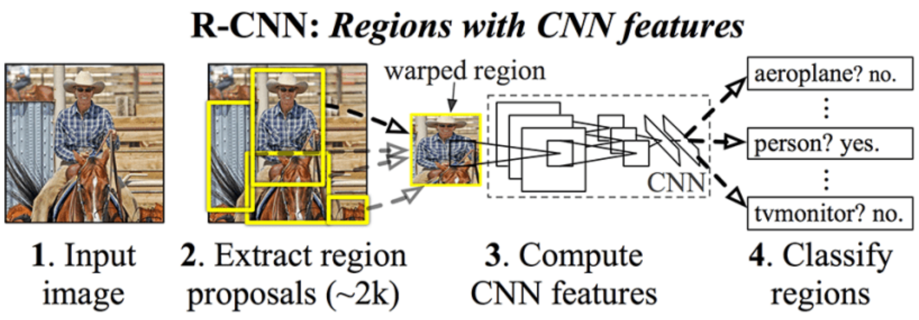


Figure 2.4. R-CNN regions working principle

2.3. FASTER R-CNN

2.3.1. Region Proposal Network (RPN)

The feature map is the RPN's input, and its output is a list of potential regions [3]. The region proposals are rectangular, as can be seen.

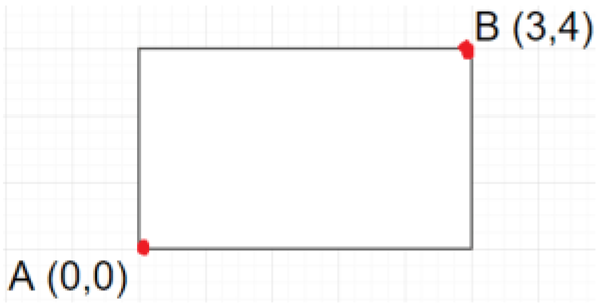


Figure 2.5. A rectangular RPN

A rectangle, for instance, is defined by two points at each of its corners, such as A (x_{min}, y_{min}) and B (x_{max}, y_{max}).

Instead of predicting the two corners, the idea is to predict the rectangle's center point (x center, y center), as well as its width and height. Thus, the x_{center}, y_{center}, width, and height parameters are used to define each anchor.

Since RPN does not employ selective search, it must first recognize anchor boxes that could be region proposals before RPN can only output anchor boxes that unquestionably contain objects.

The feature map is routed via Conv layer 3*3 and 512 kernels in the straightforward RPN model.

2.3.2. Intersection over Union (IoU)

A measurement of evaluation is Intersection over Union (IoU). Using IoU, any algorithm that outputs predicted bounding boxes can be assessed [4].

We can see in the example figure down below that our object detector has picked up on the existence of a stop sign in an image. The hand-labeled ground truth bounding box is shown in green, while the predicted bounding box is shown in red.

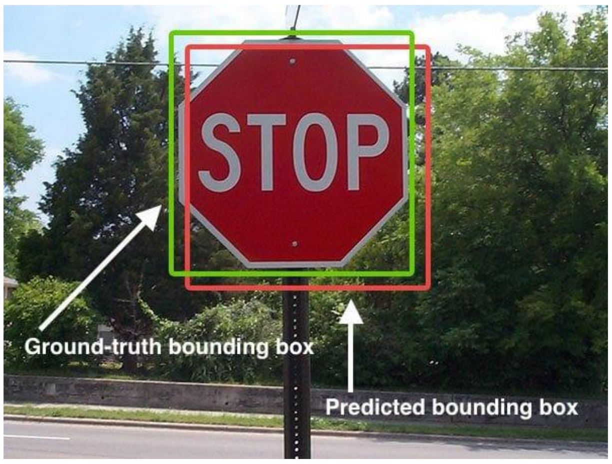


Figure 2.6. An example of detecting a stop sign in an image.

Intersection over Union can be calculate by dividing Area of Overlap to Area ôf Union

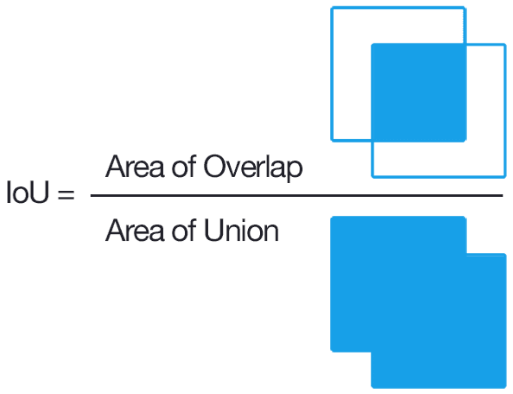


Figure 2.7. Computing Intersection over Union

A perfect and full match between predicted and ground-truth bounding boxes is just unattainable due to the variable model parameters (image pyramid scale, sliding window size, feature extraction method, etc.).

As a result, we must establish an assessment metric that favors predicted bounding boxes that closely resemble the ground truth

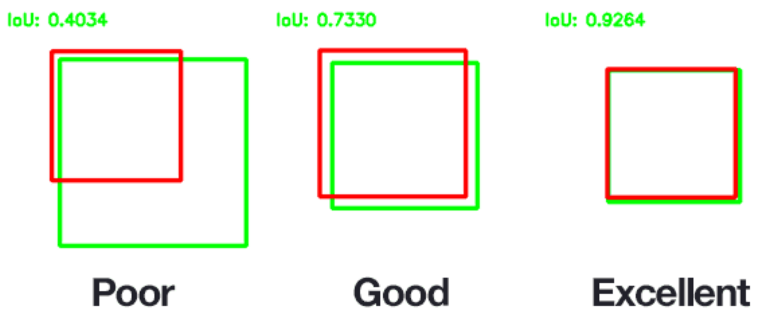


Figure 2.8. A calculation of Intersection over Unions for different bounding boxes.

Scores are greater for predicted bounding boxes that closely resemble the ground-truth bounding boxes than for those that do not. Because of this, Intersection over Union is a great statistic for measuring the performance of unique object detectors.

2.4. MASK R-CNN

Mask R-CNN was built using Faster R-CNN with 3 output of each object: class label, bounding box and object mask. The additional mask output requires the extraction of a much more precise spatial layout of an object because it differs from the class and box outputs.

Mask R-CNN has 2 main types of image segmentation: Instance Segmentation and Semantic Segmentation.

2.4.1. Semantic Segmentation

Clustering portions of an image that belong to the same object class together is known as semantic segmentation, also known as image segmentation. Since each pixel in a picture is assigned to a category, it is a type of pixel-level prediction. Pascal VOC is an example of benchmarks for this job. The Mean Intersection over Union (Mean IoU) and Pixel Accuracy metrics are typically used to assess models.

2.4.2. Object Detection

Identifying an object in an image and its location within the image frame is the job of an object detection algorithm. Bounding boxes are commonly used to define an object's location. The smallest rectangle that completely encloses an object in an image is known as a bounding box.

A bounding box technically refers to a set of four coordinates that are associated with a label that identifies the object's class. Typically, a JSON file using a dictionary format is used to store the bounding box coordinates and their labels. The dictionary file's key is the image ID or number.

CHAPTER 3: DESIGN AND IMPLEMENTATION

3.1. DATA AND MODEL

3.1.1. Datasets

Input: The dataset was included of 80 cow pictures, 80 Pascal VOC files for training, 20 cow pictures, 20 Pascal VOC files for testing.

Output: A Pascal VOC format file include bounding boxes coordinate, 1 class cow, 1 class background and 1 mask

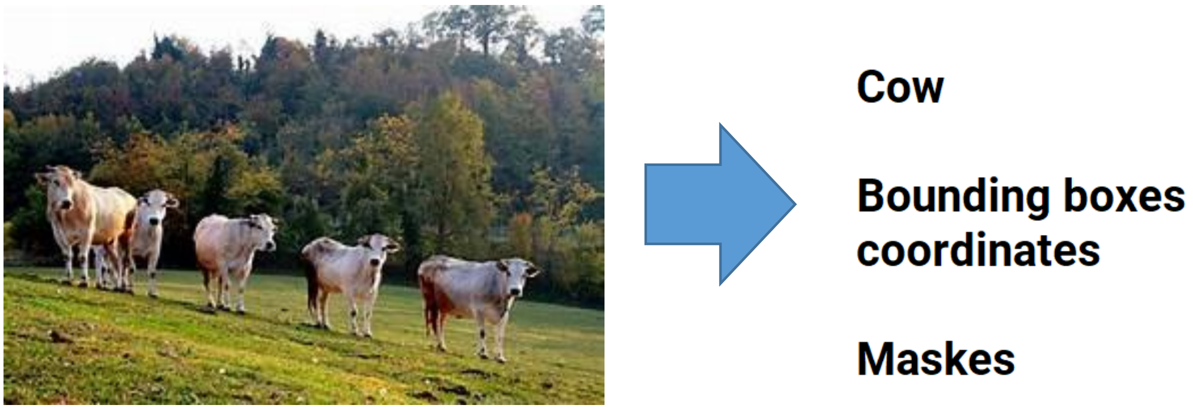


Figure 3.1. Input, output of the project.

The goal of image labeling, a type of data labeling, is to recognize and tag particular details in an image. We need to prepare 100 Pascal VOC files by using image labeling tool, which Labelimg is one of them.

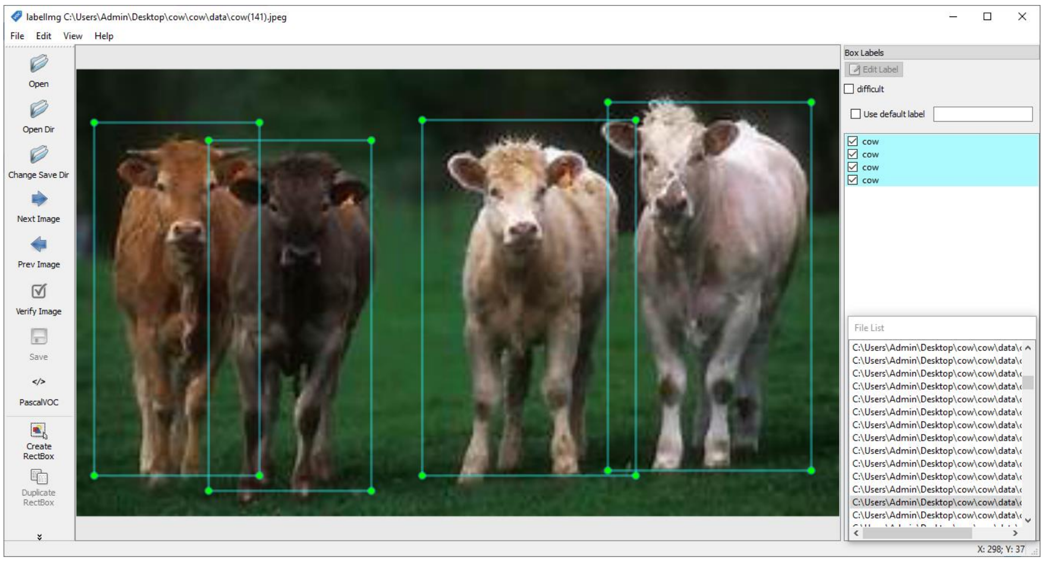
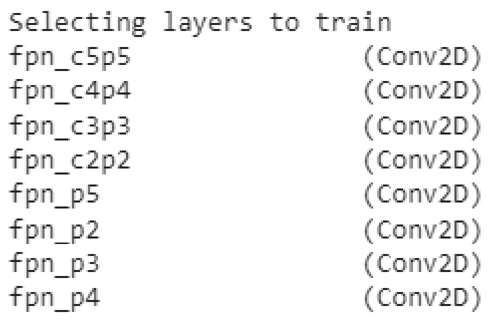


Figure 3.2. Image labeling using Labelimg

3.1.2. Model

The class was divided into 2 classes: cows and others. Each epoch need the step that equal to the training pictures. That mean the bigger datasets are, the longer it takes to train the model.

Mask R-CNN use 4 + 4 Feature Pyramid Network (FPN) layers to train the model:



```
Selecting layers to train
fpn_c5p5               (Conv2D)
fpn_c4p4               (Conv2D)
fpn_c3p3               (Conv2D)
fpn_c2p2               (Conv2D)
fpn_p5                 (Conv2D)
fpn_p2                 (Conv2D)
fpn_p3                 (Conv2D)
fpn_p4                 (Conv2D)
```

Figure 3.3. Mask R-CNN model's layers

The first 4 FPN layers fpn_cxpx only have kernel size (1x1) that is used to convert data to Conv 1x1 Mask.

On the other hand, the 4 last FPN layers fpn_px have kernel size (3x3), they are working independence to each other, included in one bottom-up pathway with decreasing resolution/ increasing semantic value, and one top-down pathway.

One FPN layer is include with one Conv2D 3x3, going through feature maps, extract it and convert data to one Conv 1x1 Regresstion, one Conv 1x1.

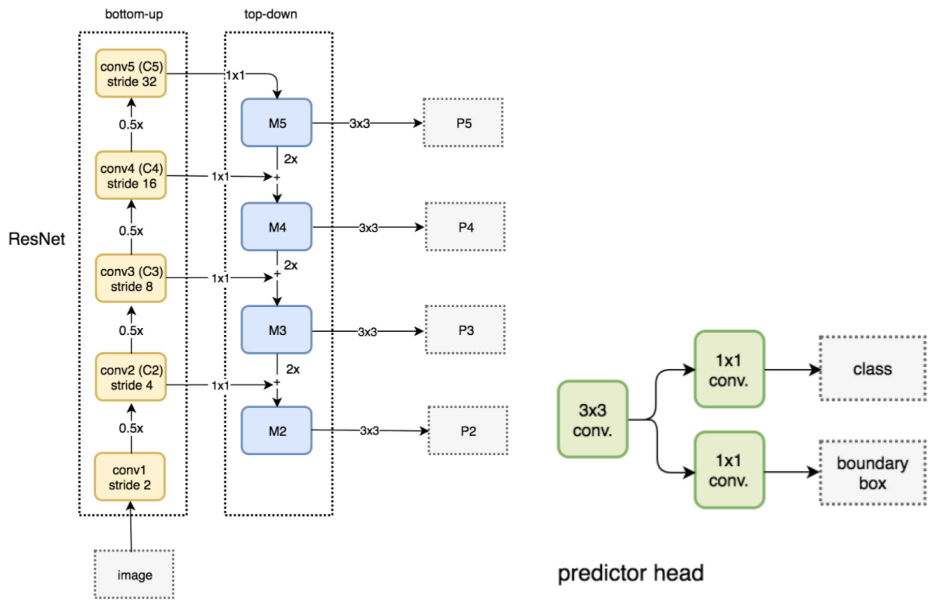


Figure 3.4. Data flow in Feature Pyramid Network

We can use tourch.nn liberary to build a similar model, define layer 2 3 4, use _upsample_add or UpSampling2D, config.TOP_DOWN_PYRAMID_SIZE, but we will use Mask R-CNN model.

Here is the full process of Mask R-CNN model training process:

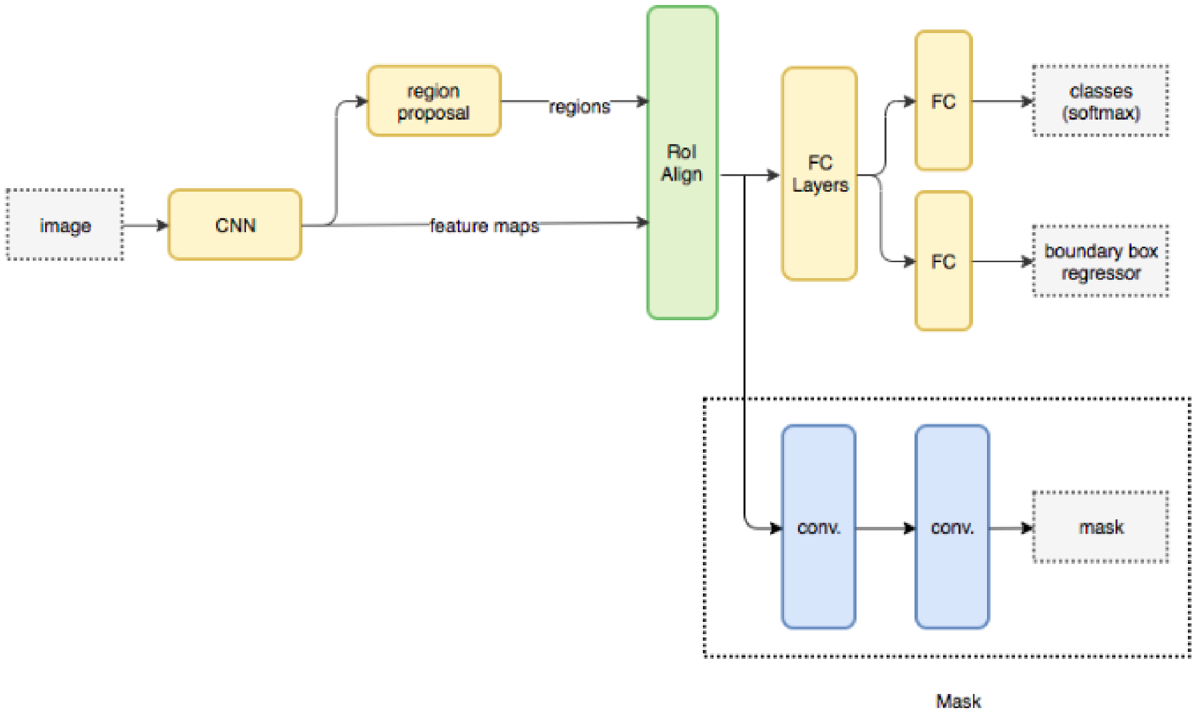


Figure 3.5. Full process of Mask R-CNN model training process

3.2. TRAINING MODEL

The training process will be in Google Colab. First, we need to install libraries.

```
!pip uninstall h5py -y
!pip install h5py==2.10.0
!pip uninstall numpy -y
!pip install numpy==1.16.4
!pip uninstall scikit-image -y
!pip install scikit-image==0.16.2
!pip install --no-deps tensorflow==1.15.3
!pip install --no-deps keras==2.2.4
```

```
 !git clone https://github.com/matterport/Mask_RCNN.git
 %cd Mask_RCNN
!pip install -r requirements.txt
!python setup.py install
%cd ..
```

Then, we import necessary libraries for training and evaluating:

```
from os import listdir
from xml.etree import ElementTree
from numpy import zeros
from numpy import asarray
from mrcnn.utils import Dataset
from mrcnn.visualize import display_instances
from mrcnn.utils import extract_bboxes
from mrcnn.config import Config
from mrcnn.model import MaskRCNN

from numpy import expand_dims
from numpy import mean
from mrcnn.utils import compute_ap
from mrcnn.model import load_image_gt
from mrcnn.model import mold_image
```

We define class CowDataset() to import prepared images and labels from Google Drive

Define load_dataset(), if is_train == True, it will load 80 first images and labels, else if is_train == False, it will load 20 later images and labels.

```
class CowDataset(Dataset):
  def load_dataset(self, dataset_dir, is_train=True):
    self.add_class("dataset", 1, "cow")
    images_dir = dataset_dir + '/data/'
    annotations_dir = dataset_dir + '/label/'
    for filename in listdir(images_dir):
      image_id = filename[4:-6]
      if is_train and int(image_id) >= 81:
        continue
      if not is_train and ( int(image_id) < 81 or int(image_id) >= 101):
        continue
      if int(image_id) >= 101:
        continue
      img_path = images_dir + filename
      ann_path = annotations_dir + 'cow(' + image_id + ').xml'
      self.add_image('dataset', image_id=image_id, path=img_path, annotation=ann_path)
```

Define extract_boxes() to read data from Pascal VOC files, then return the bounding boxes coordinate, width and height of each picture:

```
  def extract_boxes(self, filename):
      tree = ElementTree.parse(filename)
      root = tree.getroot()
      boxes = list()
      for box in root.findall('.//bndbox'):
        xmin = int(box.find('xmin').text)
        ymin = int(box.find('ymin').text)
        xmax = int(box.find('xmax').text)
        ymax = int(box.find('ymax').text)
        coors = [xmin, ymin, xmax, ymax]
        boxes.append(coors)
      width = int(root.find('.//size/width').text)
      height = int(root.find('.//size/height').text)
      return boxes, width, height
```

Define load_mask() to mask the cow using bounding box coordinate and "cow" class.

Define image_reference() to load the path of image in Google Drive:

```
  def load_mask(self, image_id):
      info = self.image_info[image_id]
      path = info['annotation']
      boxes, w, h = self.extract_boxes(path)
      masks = zeros([h, w, len(boxes)], dtype='uint8')
      class_ids = list()
      for i in range(len(boxes)):
        box = boxes[i]
        row_s, row_e = box[1], box[3]
        col_s, col_e = box[0], box[2]
        masks[row_s:row_e, col_s:col_e, i] = 1
        class_ids.append(self.class_names.index('cow'))
      return masks, asarray(class_ids, dtype='int32')

  def image_reference(self, image_id):
    info = self.image_info[image_id]
    return info['path']
```

Mount to Google Drive using command:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Load the train and test data to “train_set” and “test_set”:

```
train_set = CowDataset()
train_set.load_dataset('/content/drive/MyDrive/cow', is_train=True)
train_set.prepare()
print('Train: %d' % len(train_set.image_ids))
```

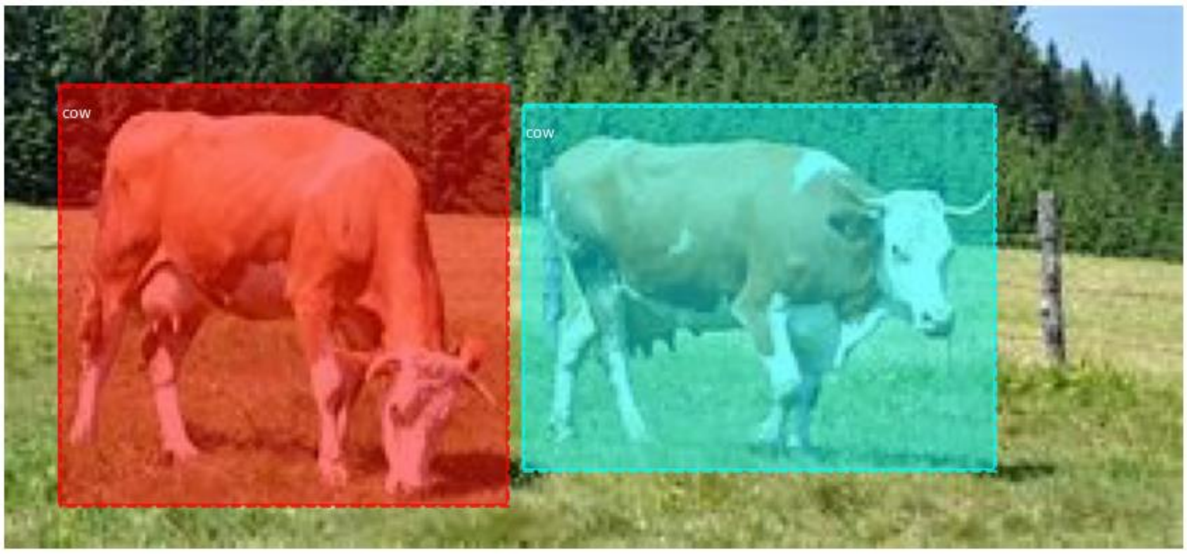
```
Train: 80
```

```
test_set = CowDataset()
test_set.load_dataset('/content/drive/MyDrive/cow', is_train=False)
test_set.prepare()
print('Test: %d' % len(test_set.image_ids))
```

```
Test: 20
```

Testing mrcnn library and the train_set data:

```
image_id = 5
image = train_set.load_image(image_id)
mask, class_ids = train_set.load_mask(image_id)
bbox = extract_bboxes(mask)
display_instances(image, bbox, mask, class_ids, train_set.class_names)
```



Define Config to auto save each epoch is done, with 2 class (cow and others) and 80 step per epoch equal to the train_set size:

```
class CowConfig(Config):
  NAME = "cow_cfg"
  NUM_CLASSES = 2
  STEPS_PER_EPOCH = 80
```

Create the model in training mode:

```
config = CowConfig()
```

```
%cd /content/drive/MyDrive
model = MaskRCNN(mode='training', model_dir='./', config=config)
```

Load mask_rcnn_coco.h5 weights and start training, using only 8 layers which is “heads”:

```
model.load_weights('mask_rcnn_coco.h5', by_name=True, exclude=["mrcnn_class_logits", "mrcnn_bbox_fc",  "mrcnn_bbox", "mrcnn_mask"])
model.train(train_set, test_set, learning_rate=config.LEARNING_RATE, epochs=5, layers='heads')
```

Even when we use Colab training consume about 2 minute each step.

```
Epoch 1/5
80/80 [==============================] - 10546s 132s/step - loss: 1.4742 -
WARNING:tensorflow:From /usr/local/lib/python3.7/dist-packages/keras/callba

Epoch 2/5
80/80 [==============================] - 10340s 129s/step - loss: 1.0119 -
Epoch 3/5
24/80 [========>.....................] - ETA: 1:40:53 - loss: 0.8643 - rpn_
```

The loss in each epoch is decreasing, but because of the method of time, we will only run 5 epochs with 0.787 loss value

```
Epoch 1/3
80/80 [==============================] - 10152s 127s/step - loss: 1.6022
WARNING:tensorflow:From /usr/local/lib/python3.7/dist-packages/keras/call

Epoch 2/3
80/80 [==============================] - 9991s 125s/step - loss: 0.8956 -
Epoch 3/3
80/80 [==============================] - 10024s 125s/step - loss: 0.7873
```

- Final result give: loss: 0.7873 - rpn_class_loss: 0.0066 - rpn_bbox_loss: 0.1696

mrcnn_class_loss: 0.105 - mrcnn_bbox_loss: 0.209 - mrcnn_mask_loss: 0.2972

val_loss: 0.9043 - val_rpn_class_loss: 0.012 - val_rpn_bbox_loss: 0.294

val_mrcnn_class_loss: 0.092 - val_mrcnn_bbox_loss: 0.215 - val_mrcnn_mask_loss: 0.29

To evaluate mAP, we need to define class with GPU count = images per GPU:

```
class PredictionConfig(Config):
  NAME = "cow_cfg"
  NUM_CLASSES = 2
  GPU_COUNT = 1
  IMAGES_PER_GPU = 1
```

```
def evaluate_model(dataset, model, cfg):
  APs = list()
  for image_id in dataset.image_ids:
    image, image_meta, gt_class_id, gt_bbox, gt_mask = load_image_gt(dataset, cfg, image_id, use_mini_mask=False)
    scaled_image = mold_image(image, cfg)
    sample = expand_dims(scaled_image, 0)
    yhat = model.detect(sample, verbose=0)
    r = yhat[0]
    AP, _, _, _ = compute_ap(gt_bbox, gt_class_id, gt_mask, r["rois"], r["class_ids"], r["scores"], r['masks'])
    APs.append(AP)
  mAP = mean(APs)
  return mAP
```

Run the evaluate_model() to train_set and test_set:

```
cfg = PredictionConfig()
%cd /content/drive/MyDrive
model = MaskRCNN(mode='inference', model_dir='./', config=cfg)
model.load_weights('mask_rcnn_cow_cfg_0003.h5', by_name=True)
train_mAP = evaluate_model(train_set, model, cfg)
print("Train mAP: %.3f" % train_mAP)
test_mAP = evaluate_model(test_set, model, cfg)
print("Test mAP: %.3f" % test_mAP)
```

The results give:

```
Train mAP: 0.715
Test mAP: 0.751
```

Which is decent enough to recognize the cows.

3.3. PREDICT RESULTS

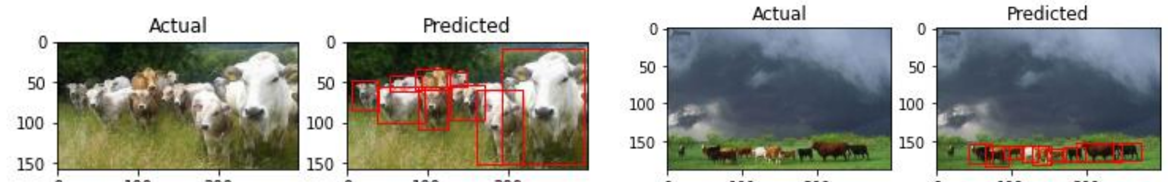
We import matplotlib library and define class:

```
def plot_actual_vs_predicted(dataset, model, cfg, i):
  image = dataset.load_image(i)
  scaled_image = mold_image(image, cfg)
  sample = expand_dims(scaled_image, 0)
  yhat = model.detect(sample, verbose=0)[0]
  pyplot.subplot(1, 2, 1)
  pyplot.imshow(image)
  pyplot.title('Actual')
  pyplot.subplot(1, 2, 2)
pyplot.imshow(image)
pyplot.title('Predicted')
ax = pyplot.gca()
biendemtam = 0
for box in yhat['rois']:
  y1, x1, y2, x2 = box
  width, height = x2 - x1, y2 - y1
  rect = Rectangle((x1, y1), width, height, fill=False, color='red')
  ax.add_patch(rect)
  biendemtam += 1
pyplot.show()
print("There are " + str(biendemtam) + " cows on this picture")
```

Then load some other image beside 100 prepared image:

```
plot_actual_vs_predicted(train_set, model, cfg, 211)
```

Here are some results from the model:

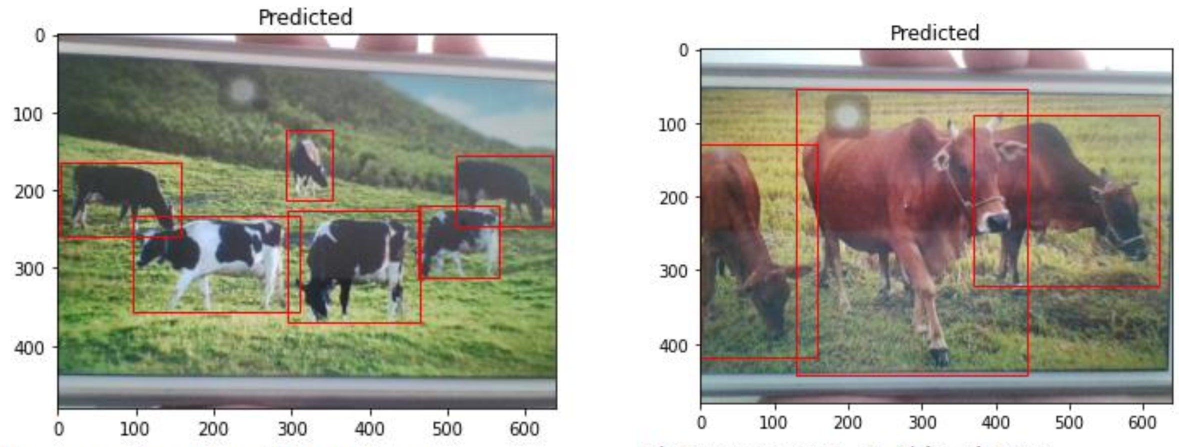


```
There are 9 cows on this picture
```

```
There are 9 cows on this picture
```

Figure 3.6. Predict results

We can also extract frame from webcam on Google Colab to run the model:



```
There are 6 cows on this picture
```

```
There are 3 cows on this picture
```

Figure 3.7. Webcam extracted frame predict results.

CHAPTER 4: CONCLUSION AND RECOMMENDATION

4.1. CONCLUSION

Since the loss value is still higher than 0.1, there will be some error in classifying, and because only 1 class of cow define mean giving more loss in value, more Intersection over Union (IoU) when classifying the head, butt and parts of other cows. But the upper results are acceptable.

In general, the model met the minimum requirements for the problem of identifying and counting cow on farms. The next step is to create more topics and draw on more experiences in order to find more cattle.

4.2. FURTHER RECOMMENDATION

The project intends to pursue the following objectives in the future:

- Rebuild and optimize the model of Mask R-CNN to decrease loss value and training time, so it can train with large dataset in a shorter amount of time.

- Define class of cows to increase the accuracy.

- Define head, butt, body of cow and solving Intersection over Union problem.

- Create enviroment datasets.

REFERENCES

[1] Uniduc JSC. "*Tìm hiểu về thuật toán R-CNN, Fast R-CNN, Faster R-CNN và Mask R-CNN*". https://uniduc.com/vi/blog/object-detection-tim-hieu-ve-thuat-toan-r-cnn-fast-r-cnn-va-faster-r-cnn. 2020.

[2] Ross Girshick Jeff Donahue Trevor Darrell Jitendra Malik. *"Rich feature hierarchies for accurate object detection and semantic segmentation".* Appendix G. Changelog. arXiv:1311.2524. 2014.

[3] Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun. "*Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*". arXiv:1506.01497v3. 2016.

[4] Adrian Rosebrock. "*Intersection over Union (IoU) for object detection*". https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/. 2016.